Avances en Arquitectura y Tecnología de Computadores

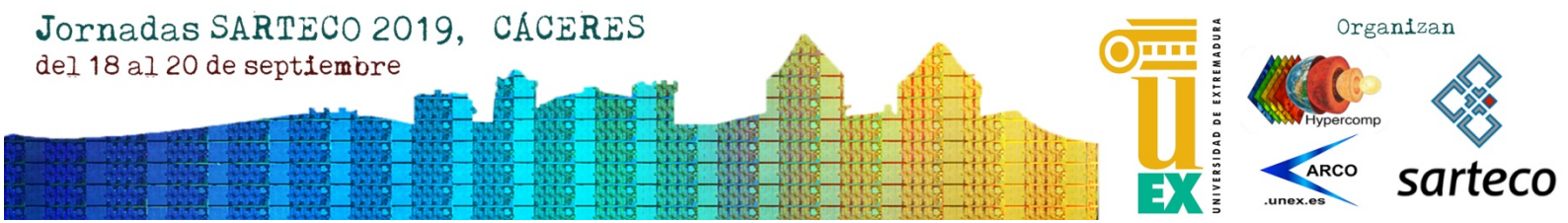
Actas de las Jornadas SARTECO 2019

Cáceres, 18 a 20 de septiembre de 2019

Editado por:

Miguel Ángel Vega Rodríguez

Antonio J. Plaza Miguel



Avances en Arquitectura y Tecnología de Computadores
Actas de las Jornadas SARTECO 2019

Editores: Miguel Ángel Vega Rodríguez y Antonio J. Plaza Miguel

Septiembre 2019, Jornadas SARTECO

ISBN-13: 978-84-09-12127-4

Cáceres, España

Publicado por:

Universidad de Extremadura. Servicio de Publicaciones

Plaza de Caldereros, 2 - Planta 3ª, 10003 Cáceres (España)

Tel.: 927 257 041; Fax: 927 257 046

Correo-e: publicac@unex.es http://www.unex.es/publicaciones

Prólogo

La Sociedad de Arquitectura y Tecnología de Computadores (SARTECO) presenta, una vez más, las Jornadas SARTECO, que integran las XXX Jornadas de Paralelismo (JP2019) y las IV Jornadas de Computación Empotrada y Reconfigurable (JCER2019). Además, en el contexto de las Jornadas SARTECO se celebra también el V Concurso "Tu Tesis en 3 Minutos" (T3M2019), que pretende premiar los mejores trabajos de tesis recientes en el área, y el III Encuentro WSARTECO de investigadoras en TIC.

Este año las Jornadas se celebran en Cáceres, contando con un excelente programa de sesiones técnicas con un total de 70 artículos que se presentarán en las JP2019 y otros 17 artículos en las JCER2019, además de 3 ponencias invitadas. Aunque la mayoría de autores de estos artículos son españoles, también hay coautores con afiliación de otros muchos países (en orden alfabético): Alemania, Argentina, Brasil, Chile, China, Ecuador, Francia, Grecia, Hungría, Italia, Japón, Marruecos, México, Perú, Portugal y Reino Unido. A fecha de edición del presente libro de actas, el número de asistentes (incluyendo conferenciantes invitados y comité de organización) es de unas 150 personas.

En conclusión, la celebración conjunta de estas actividades y eventos de carácter científico-técnico constituye un referente nacional imprescindible para la comunidad científica agrupada en SARTECO. Estas jornadas reúnen a un nutrido grupo de investigadores, procedentes de diferentes universidades y centros de investigación, con el objeto de intercambiar experiencias, presentar y debatir resultados de investigación, facilitar colaboraciones y sinergias entre grupos, y potenciar nuevas oportunidades de transferencia tecnológica a la industria.

Desde la organización de esta nueva edición de las Jornadas SARTECO, os deseamos a todos una placentera y productiva estancia en Cáceres.

Julio de 2019

Comités Organizadores

Comité de Dirección de las Jornadas SARTECO

- Inmaculada García Fernández (UMA) (Presidenta)
- Victor Viñals Yufera (UNIZAR) (Vicepresidente)
- Katzalin Olcoz Herrero (UCM) (Secretaria)
- Francisco Tirado Fernández (Presidente de Honor)

Organizadores Jornadas SARTECO2019 y Comité JP2019

- Antonio J. Plaza Miguel (UEX)
- Miguel A. Vega Rodríguez (UEX)
- Javier Plaza Miguel (UEX)
- José M. Granado Criado (UEX)
- Sergio Santander Jiménez (UEX)
- Juan M. Haut Hurtado (UEX)
- Mercedes E. Paoletti Ávila (UEX)
- Elena Paoletti Ávila (UEX)

Comité de Coordinación JCER2019

- Jesús González Peñalver (UGR)
- Sergio Cuenca Asensi (UA)
- Miguel A. Vega Rodríguez (UEX)
- Miguel Damas Hermoso (UGR)
- Antonio Martínez Álvarez (UA)
- Gustavo Sutter (UAM)
- Ignacio Bravo (UAH)
- José Torres (UV)
- Jordi Carrabina (UAB)
- Juan Suardíaz (UPCT)
- Jesús Barba Romero (UCLM)
- Goiuria Sagardui Mendieta (UMONDRAGON)
- Jorge Portilla Berrueco (UPM)

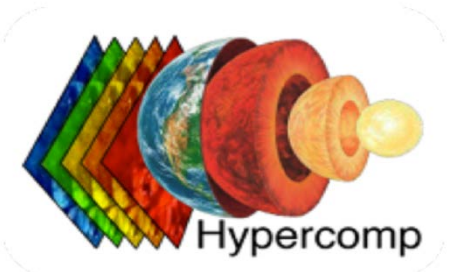
Comité de Programa JCER2019

- Jesús Barba Romero (Universidad de Castilla-La Mancha)
- Marta Beltrán (Universidad Rey Juan Carlos)
- Francisco J. Bonin-Font (Universidad de las Islas Baleares)
- Ignacio Bravo (Universidad de Alcalá)
- David Castells (Universidad Autónoma de Barcelona)
- Javier Castillo (Universidad Rey Juan Carlos)
- Sergio Cuenca Asensi (Universidad de Alicante)
- Juan Carlos Díaz Martín (Universidad de Extremadura)
- Luis Entrena (Universidad Carlos III de Madrid)
- Leire Etxeberria (Universidad de Mondragón)
- Eduard Fernández-Alonso (Recore Systems)
- Rodolfo García-Bermúdez (Universidad Técnica de Manabí, Ecuador)
- Juan Antonio Gómez Pulido (Universidad de Extremadura)
- Jesús González Peñalver (Universidad de Granada)
- José M. Granado Criado (Universidad de Extremadura)
- Juan A. Holgado-Terriza (Universidad de Granada)
- Miren Illarramendi Rezabal (Universidad de Mondragón)
- Antonio Jimeno-Morenilla (Universidad de Alicante)
- Gustavo Marrero Callico (Universidad de las Palmas de Gran Canaria)
- Antonio Martínez Álvarez (Universidad de Alicante)
- Joaquín Olivares (Universidad de Córdoba)
- Gabriel Oliver (Universidad de las Islas Baleares)
- Alberto Ortiz (Universidad de las Islas Baleares)
- Fernando Pardo (Universidad de Valencia)
- Jon Pérez (Ikerlan)
- Jorge Portilla Berrueco (Universidad Politécnica de Madrid)
- Francisco A. Pujol (Universidad de Alicante)
- Francisco Ramos (Schneider Electric)
- Lluis Ribas-Xirgo (Universidad Autónoma de Barcelona)
- José Torres (Universidad de Valencia)
- Miguel A. Vega Rodríguez (Universidad de Extremadura)
- Félix Jesús Villanueva Molina (Universidad de Castilla-La Mancha)

Comité T3M2019

- Inmaculada García Fernández (UMA) (Presidenta)
- Katzalin Olcoz Herrero (UCM) (Secretaria)
- Francisco Tirado Fernández (Presidente de Honor)
- Enrique S. Quintana Ortí (UJI) (Vocal, Junta directiva SARTECO)
- Miquel Moretó Planas (UPC) (Vocal, Junta directiva SARTECO)

Patrocinadores



Escuela Politécnica

TC2
DEPARTAMENTO DE TECNOLOGÍA
COMPUTADORES Y COMUNICACIONES

AYUNTAMIENTO
cáceres

DIPUTACIÓN DE CÁCERES

INSTITUCIÓN CULTURAL
EL BROCENSE
DIPUTACIÓN DE CÁCERES

Índice

Ponencias invitadas

Parte 1 - XXX Jornadas de Paralelismo

Aplicaciones de la computación de altas prestaciones

Redes y comunicaciones

Tecnologías clúster, plataformas distribuidas, BigData y Deep Learning

Parte 2 - IV Jornadas de Computación Empotrada y Reconfigurable

Aplicaciones y retos de la sociedad

Arquitecturas, diseños de referencia y plataformas

Conectividad de sistemas

Ponencias invitadas

Nuevas tendencias en tolerancia a fallos para aplicaciones paralelas

María José Martín Santamaría
Universidade da Coruña

Resumen

ESTUDIOS recientes demuestran que, a medida que los sistemas paralelos evolucionan, el tiempo medio hasta fallo del sistema también se reduce. Por tanto, las aplicaciones largas necesitarán incorporar algún mecanismo de tolerancia a fallos, no solamente para garantizar la finalización de sus ejecuciones, sino también para ahorrar energía. En esta conferencia discutiremos diferentes soluciones para dotar de tolerancia a fallos a las aplicaciones paralelas. Empezaremos viendo las soluciones tradicionales basadas en parada y reinicio utilizando ficheros de checkpointing, para pasar a continuación a revisar la alternativa en la que se están focalizando los esfuerzos de investigación en la actualidad, la construcción de aplicaciones resilientes.

Biografía

MARÍA J. Martín es Profesora Titular de Universidad y Directora del Departamento de Ingeniería de Computadores de la Universidade da Coruña. Tiene reconocidos 3 sexenios de investigación y está acreditada como Catedrática de Universidad desde 2015. Su principal área de investigación es la computación de altas prestaciones y, más específicamente, la computación paralela y distribuida y la tolerancia a fallos. En esta última línea ha dirigido recientemente 3 tesis doctorales centradas en la tolerancia a fallos de aplicaciones MPI, con especial énfasis en la escalabilidad. Ha sido co-autora de más de 100 artículos internacionales en el campo de la computación de altas prestaciones y ha participado de forma activa en más de 20 proyectos nacionales y regionales, siendo Investigadora Principal de uno de ellos centrado en tolerancia a fallos para aplicaciones MPI. Los resultados de su investigación han dado lugar a 4 registros software, 4 contratos con el CESGA (Centro de Supercomputación de Galicia) y 3 contratos con Hewlett-Packard para transferencia de tecnología. Ha participado en el Comité de Programa de varias conferencias internacionales como Europar y CCGRID, entre otras. También ha formado parte de la Comisión Evaluadora de Proyectos del Plan Nacional de I+D (área TIN) y del Comité Técnico de Evaluación de los contratos postdoctorales Juan de la Cierva.

From PAMELA to EuroEXA and RAIN in Manchester

Mikel Luján
University of Manchester

Resumen

THE research areas covered by the Jornadas SARTECO, namely computer architecture and systems, parallelism and reconfigurable computing, are becoming increasingly intertwined and reap from cross-fertalisation. In the PAMELA project, we started considering 3D scene understanding (also referred to as AR, VR, XR) as an important application domain that could drive heterogeneous computer systems. PAMELA has generated important insights for how to simulate and design hardware accelerators, as well as benchmarking frameworks such as SLAMBench (https://github.com/pamela-project/slambench2). The H2020 EuroEXA project is a co-design research program for HPC applications and architectures. We are investigating how reconfigurable technologies can be embraced for acceleration of computation, networking and storage operations. In the RAIN Hub, we are extending 3D scene understanding to include dynamic objects as well as considering extreme environments. This talk will cover mainly the PAMELA project, and ongoing research for the EuroEXA and the RAIN hub projects. The common thread will be the lessons that we are learning on programming and designing future heterogenous computer architectures.

Biografía

MIKEL Luján is the ARM/Royal Academy of Engineering Research Chair in Computer Systems at the University of Manchester since 2019. He is also the director of the ARM Centre of Excellence at the University of Manchester and held a Royal Society Research Fellowship since 2008 until 2017. In an industrial setting, he is the Chief Scientific Advisor of the spin off company (Amanieu Systems) which is commercialising research on Dynamic Binary Translation for modern ARM processors. Since his first paper in OOPSLA 2000, Prof. Luján has authored more than 130 refereed papers and has amassed a rich research experience extending to a range of topics from parallel programing to many-core architectures, passing by machine learning and FPGAs. For example, in the last two years he has published papers in ACM TOPLAS, IEEE TOC, ICRA, HPCA, PLDI (distinguished paper award), FCCM, VEE, and ISPASS (best paper award). In other words, he is investigating low power many-core systems considering the full stack. Mikel is well known for his contributions in speculative parallelisation, transactional memory and dynamic binary modification and translation for ARM.

Frameworks actuales para el desarrollo de la Computación Cuántica

Francisco J. Gálvez Ramírez
ABDProf Consultores

Resumen

En los últimos años grandes empresas como IBM, Google, Intel o Microsoft y otras no tan conocidas como Rigetti o Dwave han trabajado en el desarrollo de sistemas y entornos que permitan el desarrollo de programas que se ejecuten sobre ordenadores cuánticos. Actualmente existen algunos de estos ordenadores en el mercado que ofrecen empresas como IBM, Dwave o Rigetti. El próximo reto que tenemos ante nosotros es la adopción de un nuevo tipo de programación que haciendo uso de una algoritmia cuántica nos permita crear programas eficientes para este nuevo paradigma. En la conferencia veremos cuales son las últimas tendencias y los entornos que actualmente están disponibles para abordar este tipo de proyectos de programación cuántica.

Biografía

Francisco Gálvez, Licenciado en Ciencias Físicas y Máster en Física Avanzada por la Universidad de Valencia, ha trabajado 18 años en IBM y en los últimos años estuvo trabajando con la IBM Quantum Experience, difundiendo las herramientas de computación cuántica de IBM en España. Actualmente colabora con ABDProf en un proyecto de diseño e implementación de simuladores cuánticos cuyo objetivo es proporcionar entornos para el aprendizaje y la experimentación con algoritmos cuánticos.

Aplicaciones de la computación de altas prestaciones

Computación eficiente de perfiles de difusión para la extracción de información espectral-espacial

Álvaro Acción, Dora B. Heras, y Francisco Argüello[1]

Resumen— **En el ámbito del procesado de imagen multi e hiperespectral es considerado beneficioso el incluir información espacial conjuntamente con la espectral cuando se realiza el procesamiento de dichas imágenes. Los perfiles son transformaciones que extraen información espectral y espacial a diferentes niveles de granularidad en la imagen. Es posible definir perfiles basados en operaciones de difusión anisotrópica por medio de ecuaciones diferenciales parciales no lineales. Su principal ventaja es la de preservar las características morfológicas distintivas de las imágenes, como por ejemplo los bordes, en diferentes escalas. En este documento, reducimos drásticamente el alto coste computacional asociado a la construcción de perfiles de difusión para imágenes hiperespectrales mediante el uso de GPUs. En particular, proponemos un enfoque computacional de bajo coste para esta tarea empleando la arquitectura CUDA.**

I. Introducción

LAS imágenes hiperespectrales están siendo utilizadas cada vez en más contextos de aplicación debido al abaratamiento de los sensores requeridos para obtenerlas, promoviendo so uso recientemente en campos como la agricultura y la monitorización ambiental [1].

Es comúnmente aceptado en el procesamiento de imágenes hiperespectrales o multiespectrales que la extracción de información espectral-espacial mejora los resultados respecto de utilizar solamente información espectral por muy abundante que esta sea. Los perfiles morfológicos (MP) [2] son una de las técnicas más comunes utilizadas para extraer información espectral-espacial de imágenes hiperespectrales. La construcción de MPs se realiza aplicando a cada banda de la imagen transformaciones de apertura y cierre con un elemento estructural (SE) de tamaño creciente sobre las bandas obtenidas.

La difusión anisotrópica es una técnica que puede ser usada para mejorar las imágenes multiespectrales aumentando su relación señal/ruido [3] y, en particular, puede usarse para la generación de perfiles extendidos que sirven como entrada para una posterior etapa de procesado como podría ser la clasificación [4]. La aplicación del filtrado de difusión no lineal ha mostrado un gran potencial [5] en comparación con el filtrado lineal tradicional debido a su capacidad de preservar las características morfológicas distintivas de las imágenes tales como los bordes, o incluso resaltarlas.

Los esquemas utilizados para calcular la difusión no lineal presentan coste computacional elevado. La introducción de los esquemas de *Fast Explicit Diffusion* (FED) [6] ha permitido realizar estos cálculos de forma computacionalmente más eficiente, al tiempo que ofrecen una mayor precisión que los esquemas semi-implícitos existentes previamente.

Las unidades de procesamiento gráfico (GPUs) son plataformas computacionales de alto rendimiento que pueden ser usadas para el procesamiento eficiente de imágenes hiperespectrales alcanzado una ejecución en tiempo real en muchas de las aplicaciones [7], [8], [9]. La disponibilidad de GPUs de consumo capaces de realizar computación paralela permite alcanzar de forma económica cotas de rendimiento antes reservadas a sistemas como clústers o infraestructuras de altas prestaciones.

En este documento, proponemos un algoritmo programado en CUDA para tarjetas de consumo NVIDIA que realiza la extracción eficiente de información espectral-espacial en imágenes hiperespectrales. Se basa en la computación de perfiles construidos mediante filtrado realizado a través de difusión no lineal, a los que llamaremos ADPs (Anisotropic Diffusion Profiles). La unión de ADPs para diferentes bandas o componentes de la imagen producirá el EADP (Extended Anisotropic Diffusion Profile). La difusión no lineal utilizada está basada en FED, debido a su reducido coste computacional y su idoneidad para la computación paralela. Este método presenta la ventaja de que el número de componentes de perfil es bajo, lo que reduce el tiempo de ejecución de una posible clasificación posterior.

El documento está organizado en cuatro secciones. La Sección II presenta una breve introducción a los conceptos de difusión no lineal y al esquema FED utilizado para implementarla. La Sección III presenta los perfiles, sus características y la implementación en CUDA. La Sección IV contiene la evaluación de rendimiento para la clasificación propuesta sobre los conjuntos de datos de test.

II. Filtrado de difusión no lineal

La siguiente sección presentará brevemente el concepto de difusión no lineal.

La formulación clásica para la ecuación de difusión es la siguiente:

[1]Los autores pertenecen al Centro Singular de Investigación en Tecnoloxías da Información (CiTIUS), Universidade de Santiago de Compostela, España. (E-mail: alvaro.accion.montes@usc.es, dora.blanco@usc.es, francisco.arguello@usc.es)

$$\frac{\delta L}{\delta t} = \mathrm{div}(\mathrm{c(x,y,t)} \cdot \nabla \mathrm{L}). \quad (1)$$

En la ecuación anterior, div representa el operador de divergencia; ∇, el operador gradiente y c, también llamada función de conductividad, es una función que controla la difusión y la adapta a la estructura de la imagen. Perona y Malik [10] propusieron una función c variable y adaptativa que reduce el suavizado a través de los bordes y se elige en función de la magnitud del gradiente,

$$c(x,y,t) = g(|\nabla L_\sigma(x,y,t)|), \quad (2)$$

donde L_σ representa la imagen original luego de ser suavizada por el *kernel* Gaussiano de media 0 y varianza σ^2. La variable t representa de nuevo el tiempo, pero también puede considerarse como un valor de escala que controla el nivel de detalle de la imagen resultante.

Perona y Malik [10] describieron dos formulaciones diferentes para la función de conductividad, que se muestran en las ecuaciones (3) y (4).

$$g_1 = \exp\left(-\frac{|\nabla \mathrm{L}_\sigma|^2}{\mathrm{k}^2}\right), \quad (3)$$

$$g_2 = \frac{1}{1 + \frac{-|\nabla L_\sigma|^2}{k^2}}, \quad (4)$$

El parámetro k, también llamado parámetro de contraste, sirve como un umbral que permite la difusión hacia atrás. Los valores más altos de k generalmente implican que se suavicen los bordes de bajo contraste.

A. Fast Explicit Diffusion

La difusión explícita rápida (FED) es un esquema numérico eficiente que se utiliza para resolver problemas de ecuaciones parabólicas y elípticas en derivadas parciales.

FED explota el hecho de que el *kernel* Gaussiano se puede aproximar mediante cualquier *kernel* 1-D simétrico con los pesos de los coeficientes w_k que cumplen la condición $w_k = w_{-k}, \forall k \in \{1,\ldots,n\}$ y $\sum_{k=0}^{n} w_k = 1$. FED funciona realizando M ciclos compuestos por n pasos de difusión explícitos. En cada paso, se usa un τ_j variable. El valor de M controla la calidad de la aproximación, produciendo los valores más altos aproximaciones con un error más bajo a costa de una mayor complejidad.

En notación matricial, el ciclo FED se puede definir como:

$$L^{j+1} = (I + \tau_j A(L))\, L^j, \quad j = 0,\ldots,n-1, \quad (5)$$

donde L^{j+1} es la solución al problema de difusión definido por la ec. (2) en un paso dado y A es la matriz de conductividad calculada a partir de c.

Los parámetros FED se pueden expresar como,

$$\tau_j = \frac{\tau_{max}}{2\cos^2(\pi \cdot \frac{2\mathrm{j}+1}{4\mathrm{n}+2})}, \quad (6)$$

$$n = \left\lceil -\frac{1}{2} + \frac{1}{2}\sqrt{1 + \frac{12T}{M\tau_{\mathrm{máx}}}} \right\rceil, \quad (7)$$

donde τ_j es el tamaño de paso de la $j$ésima iteración resultante de la factorización del *box filter* y τ_{max} es el tamaño de paso máximo que no viola la condición de estabilidad del esquema explícito. Al conocer τ_{max}, es posible obtener el número de pasos n usando la ec. (7). Para valores de n grandes, el τ_j resultante puede ser ostensiblemente más grande que la condición de estabilidad [11].

III. Perfiles de difusión anisotrópica extendidos

En esta sección describimos la implementación en CUDA del algoritmo para la generación del perfil extendido (EADP) a partir del apilamiento de perfiles de difusión (ADPs). Se detallan también las técnicas que han permitido obtener una versión eficiente del código en CUDA, así como los resultados experimentales. Se ha analizado el rendimiento para identificar posibles cuellos de botella y decidir las optimizaciones de código que tienen un mayor impacto en el tiempo de ejecución.

Como se puede observar en la Fig. 1, cada ADP se construye aplicando un filtro de difusión no lineal a una imagen en escala de grises. En nuestro caso corresponde a una componente principal resultante de la aplicación del análisis de componentes principales (PCA) a la imagen hiperespectral original.

Una vez que se obtienen los componentes principales de la imagen, el algoritmo de difusión no lineal aplicará C instancias de difusión a cada $\boldsymbol{X}_i$ utilizando C tiempos de proceso T_i diferentes. Esta estrategia generará componentes con un nivel de detalle descendente y, por lo tanto, contendrá diferente información espacial de la imagen. El primer componente del ADP es siempre la componente principal de partida.

Finalmente, cada ADP tendrá $C+1$ componentes donde C es el número de instancias de difusión aplicadas, creando imágenes con niveles de detalle decrecientes. El EADP final tendrá un tamaño de $N \times (C+1)$, donde N es el número de componentes principales retenidos.

Denotando $\mathrm{Diff(X_i, T_i)}$ el proceso de difusión, un ADP se define como:

$$\mathrm{ADP(X_i)} = \{\mathrm{X_i}, \mathrm{Diff(X_i, T_1)}, \ldots, \mathrm{Diff(X_i, T_c)}\} \quad (8)$$

En la Fig. 2 se muestra un ejemplo de un ADP donde se puede apreciar cómo a partir de la componente inicial situada a la izquierda en la figura, el nivel de detalle se reduce de izquierda a derecha, es decir, a medida que se incorporan nuevos componentes de difusión. El EADP es el conjunto de todos

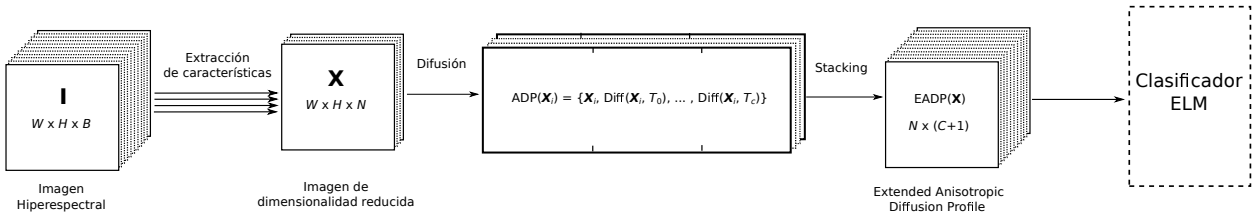


Fig. 1: Esquema propuesto para la clasificación de imágenes hiperespectrales basado en la extracción de información espacial mediante el uso de EADP.

los ADP resultantes de la aplicación del proceso de difusión:

$$\mathrm{EADP}(\mathbf{X}) = \{\mathrm{ADP}(\mathrm{X}_1), ..., \mathrm{ADP}(\mathrm{X}_\mathrm{N})\} \quad (9)$$

Los requisitos de memoria del algoritmo que construye el EADP dependen solo de las dimensiones de la imagen hiperespectral que se procesa y del número de componentes después de la etapa de extracción de características. El uso de memoria permanece constante durante toda la ejecución y requiere 4 *buffers*. La cantidad de memoria requerida es de aproximadamente $4WHN$ palabras.

A. Implementación en GPU

CUDA es un modelo de programación y arquitectura de computación en paralelo desarrollado por NVIDIA que permite la ejecución de funciones relativamente simples, llamadas *kernels*, dentro de una GPU. Se ha desarrollado como primera versión un código para ser ejecutado en una única GPU de consumo.

Para conseguir un código eficiente en CUDA se han aplicado múltiples técnicas de optimización. Dichas técnicas están basadas en una gran parte en optimizar el uso de la jerarquía de memoria: haciendo cálculos *in-place*, tratando de fomentar la reutilización de datos, usando memoria *pinned* para las transferencias entre CPU y GPU, o usando operaciones atómicas en memoria compartida frente a realizarlas en memoria global debido al soporte específico para operaciones atómicas existente desde la arquitectura Maxwell.

Otra técnica para evitar la latencia de acceso a la memoria compartida es realizar operaciones de reducción utilizando primitivas a nivel de *warp*. Se han utilizado también tipos de datos vectoriales para maximizar el paralelismo de instrucciones y optimizar los accesos a memoria. Finalmente, se han usado librerías de alto rendimiento como cuBLAS, cuSOLVER y cuSPARSE para la realización de algunas operaciones.

Tal como se muestra en el pseudocódigo presentado en el algoritmo 1, tras una etapa de extracción de características, que en este caso se ha realizado en GPU mediante el algoritmo PCA [12], se ha procedido al cálculo de la difusividad.

El cálculo de la matriz comienza con la carga inicial de $\boldsymbol{X}_i$ en la memoria de la GPU. Una vez cargada la componente, se realizan dos convoluciones con un

Algorithm 1 Pseudocódigo la construcción de EADP en CUDA

Entrada:
$\boldsymbol{X}_i$: Componente principal, $i = 1..N$.
σ^2: Varianza del filtro Gaussiano.
$\{T_1, \ldots, T_c\}$: lista de tiempos de proceso.
Salida:
EADP($\mathbf{X}$): EADP de la imagen $\mathbf{X}$.

```
1:  EADP ← ∅
2:  for i=1 →N do
3:      ADP_i ← ∅
        Cálculo de matriz de difusividad
4:      X_{i,σ} ← < apply_row_conv >(X_i, G_σ(X_i))        ▷ SM+GM
5:      X_{i,σ} ← < apply_col_conv >(X_{i,σ}, G_σ(X_i))    ▷ SM+GM
6:      ∇X_{i,σ} ← < scharr >(X_{i,σ}, Scharr(X_{i,σ}))    ▷ SM+GM
7:      max ← < reduce_max >(X_{i,σ})                      ▷ REG+GM
8:      histo ← < create_histogram >(∇X_{i,σ}, max)        ▷ SM+GM
9:      k ← get_bin(histo, 0,7)                            ▷ SM+GM
10:     A(∇X_{i,σ}) ← < pm2_coefficients >(∇X_{i,σ}, k)    ▷ GM
        Cálculo de proceso FED
11:     X_i^0 ← X_i
12:     for c = 1 → C do
13:         τ_j → fed_tau_by_process_time(T_c)
14:         X_i^{c,0} ← X_i^{c−1}
15:         for j = 1 → n do
16:             ΔX_i^{c,j} ← < fed_nld_step >(X_i^{c,j−1}, A(∇X_{i,σ}))
                                                           ▷ SM+GM
17:             X_i^{c+1,j+1} ← < fed_nld_update >(X_i^{c,j}, ΔX_i^{c,j})
                                                           ▷ GM
18:         end for
19:         ADP_c ← X^{c,n} ∪ ADP_c
20:     end for
21:     EADP ← ADP_c ∪ EADP
22: end for
```

kernel Gaussiano para suavizar la imagen original (líneas 4-5 en el pseudocódigo). Cada convolución es realizada por un *kernel* específicamente diseñado para explotar la localidad de la memoria dependiendo de la dirección de la operación. A continuación, el *kernel* que calcula las derivadas de la imagen obtiene $\nabla \boldsymbol{X}_{i,\sigma}$ (línea 6). Después de eso, el valor máximo de la imagen suavizada, $G_\sigma(\boldsymbol{X}_i)$, se obtiene con un *kernel* de reducción (línea 7). Dicho máximo se usa en la creación de un histograma.

Un nuevo *kernel* tomará $\nabla \boldsymbol{X}_{i,\sigma}$ y computará un histograma con las distancias entre las derivadas vertical y horizontal (línea 8), almacenándolas atómicamente en la memoria compartida y luego sumándolos atómicamente al histograma en la memoria global. El parámetro de contraste se calcula en CPU (línea 9), debido a la baja complejidad computacional de la tarea. Por último, la matriz de difusividad se calcula aplicando la ec. (4) a cada píxel de la imagen (línea 10).

El cálculo inicial del número de pasos y τ_j se ejecutan en la CPU (línea 13). El proceso de difusión

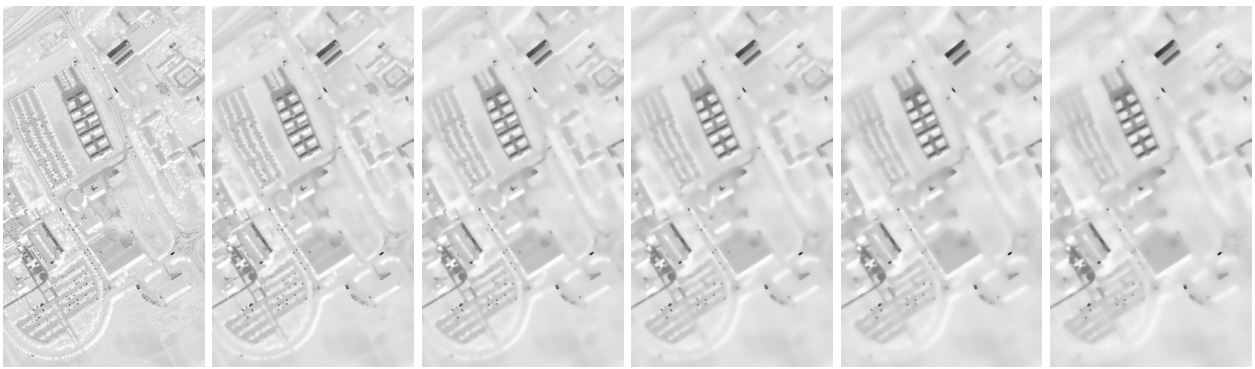


Fig. 2: La imagen representa un ADP para $N = 1$, $C = 5$ y de izquierda a derecha, una componente principal de la imagen seguida de las componentes resultantes de la difusión.

realiza la actualización de $\boldsymbol{X}_i^{c+1,j+1}$ de forma iterativa dentro de un *kernel* (líneas 16-17). Por último se añade cada componente al ADP correspondiente y este al EADP global (líneas 18-21).

IV. Evaluación del rendimiento

Esta sección describe los conjuntos de datos utilizados, explica los parámetros seleccionados para las pruebas y analiza la calidad de la propuesta presentada. Por un lado, se analiza el tiempo de ejecución en comparación con una difusión computada utilizando solamente la CPU multinúcleo de los experimentos. Por otro lado, para demostrar la efectividad del enfoque propuesto, incluimos un análisis de precisión de la clasificación obtenida utilizando el EADP comparando con las técnicas más comunes en la literatura.

Para realizar los experimentos que permiten evaluar esta técnica se han utilizado cuatro imágenes hiperespectrales reales. Por un lado, Indian Pines (IndianP) y Salinas Valley (Salinas), ambas tomadas por el sensor AVIRIS de la NASA. La resolución espacial es de 20 metros/píxel y cubre un rango espectral de 400 a 2500 nm, con 220 bandas espectrales en total. Los datos de referencia disponibles para cada imagen se dividen en dieciséis clases. IndianP y Salinas tienen unas dimensiones de 145 x 145 y 512 x 217 píxeles respectivamente. Por otro lado, Pavia University (PaviaU) y Pavia Center (PaviaC), adquiridas por el sensor ROSIS-03 en la ciudad de Pavia, Italia. Su resolución espacial es de 2,6 metros/píxel y cubre el rango espectral de 430 a 860 nm, con un total de 103 bandas espectrales. Los datos de referencia contienen nueve clases.Las dimensiones espaciales de PaviaU son 610 x 340 píxeles y las de PaviaC de 610 x 610 píxeles.

Los experimentos se llevaron a cabo utilizando una estación de trabajo con CPU Intel Core i5 8400 de 6 núcleos que funciona a 2,80 GHz y 32 GB de RAM. También dispone de una GPU NVIDIA GeForce GTX 1060 de 6 GB. Todos los experimentos se ejecutaron bajo Ubuntu Linux 16.04 de 64 bits y se compilaron con GCC versión 6.4.0 y CUDA toolkit 9.1.

Respecto del esquema FED se seleccionaron los valores de parámetros que respetasen un compromiso razonable entre precisión y velocidad, siendo estos valores $M = 1$, $\tau_{max} = 0{,}25$. Para la clasificación se ha utilizado el clasificador ELM detallado en [13]. La ejecución de ELM requiere fijar el número de muestras seleccionadas de cada clase para entrenar, que fijamos a 200 y el número de neuronas de la capa oculta que ha sido de 250 neuronas.

TABLA I: Comparativa en términos de clasificación con perfiles morfológicos

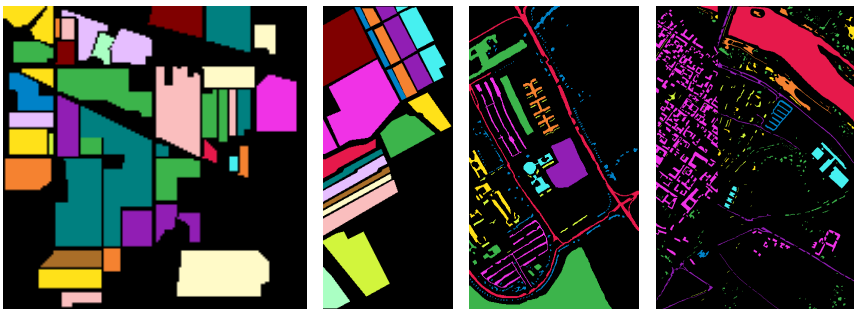
Dataset	Método	**OA**	$\boldsymbol{\kappa}$
Salinas	PCA-EADP	98.40	98.25
	WT-EMP [14]	91.67	90.70
	WTSS-EMP [14]	98.44	98.30
PaviaC	PCA-EADP	99.20	98.85
	WT-EMP [14]	99.54	99.30
	WTSS-EMP [14]	99.86	99.80

A. Precisión de la clasificación

Para evaluar la precisión de la clasificación, se consideraron las medidas estándar: precisión general medida como porcentaje de píxeles en los que se produce acierto en la clasificación (OA) y coeficiente kappa (κ).

Puede observarse, tal y como mostramos en la Fig. 4, que el valor de OA observado al clasificar una imagen usando perfiles de difusión aumenta con el número de componentes principales extraídas de la imagen para un paso de tiempo de proceso fijo. Sin embargo, el aumento del tamaño del perfil que supone este aumento en número de componentes acarrea un mayor uso de recursos en la posterior clasificación, por lo que finalmente se optó por seleccionar para los restantes experimentos 7 componentes principales ($N = 7$). Los valores de los parámetros $N = 7$, $C = 8$, $\sigma = 1$ y $ts = 65$ se utilizaron como un compromiso entre la precisión y el número de componentes del EADP. El tamaño total de cada ADP es, por lo tanto, 9 y el tamaño total de EADP es 63.

La tabla I compara la precisión de la clasificación en términos de OA y $\boldsymbol{\kappa}$ con otras dos propuestas diferentes para extraer información espectral-espacial



(a) (b) (c) (d)

Fig. 3: Imágenes hiperespectrales utilizadas en los experimientos: (a) IndianP, (b) Salinas, (c) PaviaU, (d) PaviaC

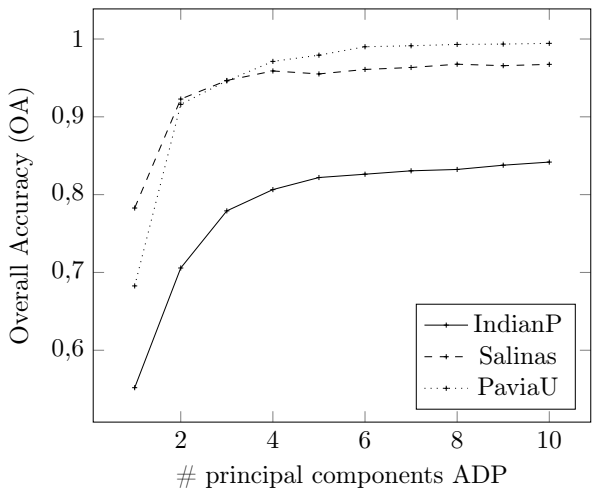


Fig. 4: OA variando el número de componentes principales extraídas para un ADP con ts=5.

que hemos desarrollado en trabajos anteriores. La primera es WT-EMP [15] y consiste en extraer características de la imagen original mediante una wavelet de modo similar a cómo en esta propuesta lo hacemos con PCA, construyendo luego un perfil morfológico extendido al que se le aplica reducción de ruido también mediante wavelets. En el caso de WTSS-EMP [14] el modelo anterior se refina añadiendo al resultado descrito una versión de la imagen original donde se ha eliminado ruido banda a banda. Los resultados muestran que los perfiles de difusión mejoran sustancialmente los resultados anteriores ya que extraen más eficientemente la información espacial.

B. Comparativa de rendimiento de CUDA

Con el fin de evaluar el rendimiento de la implementación en CUDA tomando como referencia una implementación eficiente en CPU, se desarrolló una implementación OpenMP del algoritmo que utiliza los 6 núcleos disponibles.

La Tabla II muestra una comparativa de rendimiento entre la CPU y las implementaciones de GPU. El proceso de difusión se divide en etapas y se muestra el tiempo total agregado (en milisegundos) para todas las iteraciones en cada etapa, así como la aceleración del código CUDA sobre el OpenMP.

Las etapas de *Setup* y *Cleanup* corresponden a la carga inicial del componente principal en la memoria del dispositivo y la transferencia de la imagen resultado del proceso de difusión a la memoria del host (CPU), respectivamente, que resultan ser las más costosas en la implementación en GPU. Las otras etapas hacen referencia a las líneas detalladas en el algoritmo 1.

Se puede observar que el aumento de rendimiento de la implementación en GPU es notable en las etapas de computación. La ganancia máxima se puede observar en el dataset IndianP, con un 10,47× de speedup sobre la implementación en OpenMP.

Podemos comprobar que la implementación de CUDA supera a la CPU en casi un orden de magnitud. Las aceleraciones logradas no muestran correlación con el tamaño de la imagen, con todas las escenas produciendo resultados similares. Es importante recalcar que, debido a los requisitos del algoritmo de difusión, las operaciones deben realizarse con aritmética de doble precisión, que está severamente limitada en las GPU NVIDIA de consumo.

V. Conclusiones

En este artículo se presenta una primera implementación en CUDA de perfiles de difusión anisotrópica (ADP) para extraer información espectral-espacial en imágenes hiperespectrales. Estos se crean aplicando múltiples instancias de difusión no lineal a componentes extraídas mediante análisis en componentes principales (PCA), que posteriormente se apilan para generar un perfil extendido (EADP). La implementación propuesta alcanza un rendimiento de hasta 10,47× para IndianP en comparación con la implementación OpenMP de referencia.

El reducido número de componentes del EADP permite un proceso de clasificación rápido y eficiente utilizando un algoritmo de clasificación supervisada como ELM, alcanzando valores de precisión de clasificación de hasta OA de 99,20 para PaviaC.

Agradecimientos

Este trabajo fue apoyado en parte por la Consellería de Educación, Universidade e Formación Profesional bajo las Subvenciones GRC2014/008, ED431C 2018/2019 y ED431G/08 y el Ministerio de Econo-

TABLA II: Tiempo medio de ejecución OpenMP y CUDA (en milisegundos) para la generación de un EADP de 63 componentes (a partir de 7 componentes principales obtenidas mediante PCA).

Etapa (líneas)	IndianP			Salinas			PaviaU		
	CPU	GPU	Speedup	CPU	GPU	Speedup	CPU	GPU	Speedup
Setup	0.001	0.021	0.005×	0.001	0.08	0.009×	0.001	0.137	0.007×
Matriz de difusividad	3.88	0.33	11.79×	9.73	0.59	16.55×	21.29	0.55	38.64×
Proceso FED	7.58	0.72	10.48×	27.15	3.26	8.32×	41.76	5.31	7.86×
Cleanup	0.001	0.029	0.05×	0.002	0.08	0.24×	0.002	0.137	0.02×
Total	11.46	1.09	10.47×	36.88	4.01	9.21×	63.06	6.14	10.27×

mía y Empresa, Gobierno de España bajo la subvención TIN2016-76373-P. Ambos están cofinanciados por el Fondo Europeo de Desarrollo Regional.

Referencias

[1] Gary A Shaw, "Spectral imaging for remote sensing," *Lincoln Laboratory Journal*, vol. 14, no. 1, pp. 3–28, 2003.

[2] Mathieu Fauvel, Jón Atli Benediktsson, Jocelyn Chanussot, and Johannes R Sveinsson, "Spectral and spatial classification of hyperspectral data using SVMs and morphological profiles," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 46, no. 11, pp. 3804–3814, 2008.

[3] ST Acton and J Landis, "Multi-spectral anisotropic diffusion," *International Journal of Remote Sensing*, vol. 18, no. 13, pp. 2877–2886, 1997.

[4] Fardin Mirzapour and Hassan Ghassemian, "Hyperspectral image classification using profiles based on partial differential equations," in *Electrical Engineering (ICEE), 2015 23rd Iranian Conference on*. IEEE, 2015, pp. 288–292.

[5] Yi Wang, Ruiqing Niu, and Xin Yu, "Anisotropic diffusion for hyperspectral imagery enhancement," *IEEE Sensors Journal*, vol. 10, no. 3, pp. 469–477, 2010.

[6] Sven Grewenig, Joachim Weickert, and Andrés Bruhn, "From box filtering to fast explicit diffusion," in *Joint Pattern Recognition Symposium*. Springer, 2010, pp. 533–542.

[7] Álvaro Ordóñez, Francisco Argüello, and Dora B Heras, "GPU accelerated FFT-based registration of hyperspectral scenes," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 10, no. 11, pp. 4869–4878, 2017.

[8] Yan Ma, Lajiao Chen, Peng Liu, and Ke Lu, "Parallel programing templates for remote sensing image processing on GPU architectures: design and implementation," *Computing*, vol. 98, no. 1-2, pp. 7–33, 2016.

[9] Sergio Bernabe, Sergio Sanchez, Antonio Plaza, Sebastián López, Jón Atli Benediktsson, and Roberto Sarmiento, "Hyperspectral unmixing on gpus and multi-core processors: A comparison," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 6, no. 3, pp. 1386–1398, 2013.

[10] Pietro Perona and Jitendra Malik, "Scale-space and edge detection using anisotropic diffusion," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 12, no. 7, pp. 629–639, 1990.

[11] Pascal Gwosdek, Sven Grewenig, Andrés Bruhn, and Joachim Weickert, "Theoretical foundations of gaussian convolution by extended box filtering," in *International Conference on Scale Space and Variational Methods in Computer Vision*. Springer, 2011, pp. 447–458.

[12] Alberto S Garea, Dora B Heras, and Francisco Argüello, "GPU classification of remote-sensing images using kernel ELM and extended morphological profiles," *International Journal of Remote Sensing*, vol. 37, no. 24, pp. 5918–5935, 2016.

[13] Javier López-Fandiño, Pablo Quesada-Barriuso, Dora B Heras, and Francisco Argüello, "Efficient ELM-based techniques for the classification of hyperspectral remote sensing images on commodity GPUs," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 8, no. 6, pp. 2884–2893, 2015.

[14] Pablo Quesada-Barriuso, Dora B Heras, and Francisco Argüello, "Exploring the impact of wavelet-based denoising in the classification of remote sensing hyperspectral images," in *Image and Signal Processing for Remote Sensing XXII*. International Society for Optics and Photonics, 2016, vol. 10004, p. 100040R.

[15] Pablo Quesada-Barriuso, Francisco Argüello, Dora B Heras, and Jón Atli Benediktsson, "Wavelet-based classification of hyperspectral images using extended morphological profiles on graphics processing units," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 8, no. 6, pp. 2962–2970, 2015.

Taxonomía de Algoritmos Evolutivos Multiobjetivo Paralelos: Una Visión Intra-Algorítmica

Sergio Santander-Jiménez[1] y Miguel A. Vega-Rodríguez[2]

Resumen— **El paralelismo se ha convertido en una herramienta fundamental para complementar a la computación evolutiva en la resolución de problemas de optimización reales, especialmente en el contexto multiobjetivo. Sin embargo, la selección de diseños evolutivos paralelos representa una cuestión compleja debido a las múltiples variables que se deben considerar para obtener una calidad de soluciones adecuada así como una explotación precisa de los recursos hardware. Este trabajo trata de investigar estos factores a través de un análisis comparativo de diseños paralelos intra-algoritmo en configuraciones de memoria compartida. Para ello, se han considerado distintas tendencias taxonómicas de diseño incluyendo A) aproximaciones generacionales basadas en medidas de calidad de solución y diversidad, B) aproximaciones generacionales basadas en medidas de calidad de solución exclusivamente y C) aproximaciones no generacionales. Los experimentos realizados en escenarios de optimización reales dan cuenta de las ventajas e inconvenientes de cada diseño, pudiéndose definir guías para la selección de métodos según las características del hardware, propiedades evolutivas y la capacidad de explotación del paralelismo de cada aproximación.**

Palabras clave— **Taxonomía, Paralelismo, Optimización Multiobjetivo, Algoritmos Evolutivos.**

I. Introducción

La necesidad de resolver problemas de optimización NP-completos juega un papel predominante en las líneas de investigación de múltiples dominios científicos. La complejidad de estos problemas, especialmente cuando es preciso optimizar varias funciones objetivo simultáneamente, justifica los esfuerzos en desarrollar métodos estocásticos de resolución. En este contexto, resultan de especial relevancia los resultados obtenidos mediante el uso de algoritmos evolutivos multiobjetivo (o MOEAs, del inglés *Multi-Objective Evolutionary Algorithms*) [1]. A pesar de ello, los problemas de optimización reales están gobernados por la presencia de distintos factores que afectan a su dificultad, incluyendo espacios de búsqueda exponencialmente crecientes, alta dimensionalidad y funciones objetivo de elevado tiempo de ejecución. Como resultado, el empleo de algoritmos serie no permite satisfacer los requisitos temporales asociados a estos problemas en la actualidad.

Los avances en desarrollo hardware abren la puerta a la resolución eficiente de problemas de optimización mediante la explotación del paralelismo presente en los métodos evolutivos. De hecho, la combinación de paralelismo y computación evolutiva es conocida por proporcionar una serie de beneficios, incluyendo tiempos de procesamiento reducidos, mayor robustez en distintas instancias del problema y mejora en la calidad de las soluciones. La Figura 1 muestra una representación esquemática de los distintos tipos de MOEAs paralelos, según el enfoque con el que el paralelismo es aplicado [2]. Por un lado, las aproximaciones inter-algoritmo emplean paralelismo con el objetivo fundamental de mejorar la calidad de las soluciones encontradas, combinando en el mismo lapso temporal la acción conjunta de distintos procesadores organizados mediante esquemas de grano grueso (islas) o grano fino (difusión). Por otra parte, las aproximaciones intra-algoritmo aplican paralelismo para minimizar el tiempo de ejecución y acelerar el proceso de optimización, paralelizando funciones costosas dependientes de la formulación del problema y/o aquellas propias del MOEA mediante esquemas maestro-esclavo y de compartición del trabajo.

Dadas las propiedades computacionalmente costosas de los problemas multiobjetivo actuales, este trabajo se focaliza en el estudio de aproximaciones de paralelización intra-algoritmo. Concretamente, nos centraremos en diseños que aplican paralelismo sobre los bucles propios del algoritmo evolutivo (independiente del problema) y donde no existe separación entre funciones objetivo (cada procesador evalúa todos los objetivos sobre los individuos asignados). El rendimiento de estos diseños se ve afectado no solo por requisitos relacionados con el problema abordado, sino también por las características del hardware y, especialmente, por el paralelismo disponible intrínseco de cada MOEA. De hecho, una cuestión clave radica en cuantificar el impacto de los diseños algorítmicos de cada MOEA en las ganancias temporales obtenidas por sus versiones paralelas, con objeto de determinar el MOEA paralelo que mejor se ajuste a las preferencias del experto en términos de calidad de solución y tiempo de ejecución.

Con este propósito, este trabajo define y estudia distintos componentes taxonómicos intra-algoritmo de memoria compartida, con vistas a ayudar a los investigadores en la elección de métodos multiobjetivo de optimización según las características del hardware disponible. Evaluaremos el rendimiento paralelo y multiobjetivo obtenidos por diseños paralelos: 1) generacionales que incorporan medidas de calidad

[1]Instituto de Engenharia de Sistemas e Computadores - Investigação e Desenvolvimento em Lisboa (INESC-ID), Instituto Superior Técnico, Universidade de Lisboa, Lisboa 1000-029, Portugal, e-mail: sergio.jimenez@tecnico.ulisboa.pt.

[2]Universidad de Extremadura, Departamento de Tecnología de los Computadores y de las Comunicaciones, Escuela Politécnica. Campus Universitario s/n, Cáceres 10003, España, e-mail: mavega@unex.es.

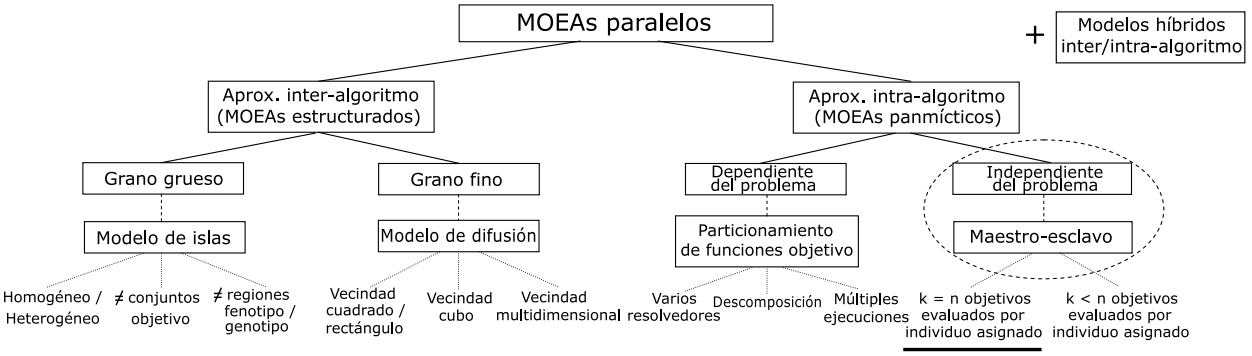


Fig. 1: Clasificación general de MOEAs paralelos, destacando el enfoque de este trabajo

de solución y diversidad (representados por el *Non-dominated Sorting Genetic Algorithm II*, NSGA-II [3], *Strength Pareto Evolutionary Algorithm 2*, SPEA2 [4], y el *Indicator-Based Evolutionary Algorithm*, IBEA [5])), 2) generacionales basados en medidas de calidad de solución exclusivamente (*Multi-objective Firefly Algorithm*, MO-FA [6], y *Multiobjective Evolutionary Algorithm Based on Decomposition*, MOEA/D [7]) y 3) no generacionales (*Multiobjective Artificial Bee Colony*, MOABC [8]).

Este artículo se organiza del siguiente modo. La siguiente sección describe los MOEA paralelos considerados, mientras que la Sección III define el escenario de optimización real abordado como benchmark. La evaluación y discusión de los MOEAs paralelos es detallada en la Sección IV. Finalmente, la Sección V incluye conclusiones y líneas de trabajo futuro.

II. MOEAs Paralelos

En esta sección, describimos los diseños de MOEAs paralelos intra-algoritmo considerados, categorizados conforme a lo representado en la Figura 2. Al centrarse este análisis en plataformas de memoria compartida, se ha empleado el estándar OpenMP [9] en la implementación de los algoritmos evaluados.

A. Diseños Generacionales

Los diseños generacionales involucran la distribución de las operaciones susceptibles de paralelización que conforman cada iteración (generación) del MOEA. Solo cuando todas las tareas paralelas pertenecientes a una generación han terminado, el algoritmo puede proceder con la siguiente, por lo que estos diseños están asociados a estilos síncronos de programación paralela. Evaluaremos dos tendencias: aproximaciones basadas en medidas de calidad de solución y diversidad y aproximaciones basadas en medidas de calidad de solución exclusivamente.

A.1 Calidad de Solución y Diversidad

Estos diseños se caracterizan por incluir mecanismos de evaluación de soluciones conforme a criterios de convergencia y diversidad. En nuestro análisis, estas aproximaciones vienen representadas por tres MOEAs: NSGA-II [3], SPEA2 [4] e IBEA [5]. Estos algoritmos emplean el esquema evolutivo tradicional, generando nuevas soluciones mediante selección, cruce y mutación. La principal diferencia radica en las estrategias de fitness y actualización de población implementadas. Por un lado, NSGA-II incluye ordenación rápida no dominada y cálculo de distancias de *crowding* para determinar la población padre de la siguiente generación. Por su parte, SPEA 2 emplea un archivo de individuos prometedores que es actualizado mediante valores de *strength* y medidas de diversidad basadas en el vecino más próximo. Finalmente, IBEA calcula valores de fitness en base a indicadores de calidad multiobjetivo (por ejemplo, hipervolumen) para identificar los individuos que deben permanecer en la población e intervenir en el cálculo de nuevas soluciones candidatas.

El Algoritmo 1 presenta el esquema básico paralelo de NSGA-II, SPEA2 e IBEA. En estos diseños,

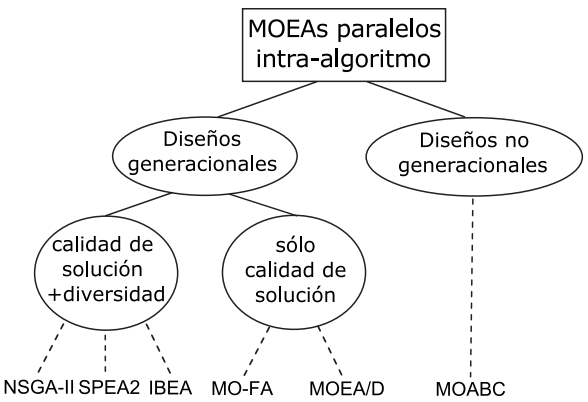


Fig. 2: MOEAs paralelos estudiados en este trabajo

Algoritmo 1 NSGA-II/SPEA2/IBEA paralelo generacional

```
1:  Inicializar Estructuras de Datos (EstPoblación, Frente)
2:  #pragma omp parallel num_threads (numHilos)
3:  Inicializar Población (EstPoblación, tamañoPob, numHilos)
4:  mientras ! criterio de parada hacer
5:     #pragma omp single
6:        Cálculo de Fitness Multiobjetivo (EstPoblación)
7:        Actualización Población Padre (EstPoblación)
8:     #pragma omp for schedule (tipoPlanificación)
9:     para i = 1 hasta tamañoPob hacer
10:       Selección, cruce y mutación (P'_i, EstPoblación, probCruce, probMutación)
11:       Evaluar Solución Generada (P'_i)
12:    fin para
13:    #pragma omp single
14:       Actualizar Frente Pareto (Frente, EstPoblación∪P')
15: fin mientras
```

Algoritmo 2 MO-FA paralelo generacional

```
1:  Inicializar Estructuras de Datos (P, Frente)
2:  #pragma omp parallel num_threads (numHilos)
3:  Inicializar Población (P, tamañoPob, numHilos)
4:  mientras ! criterio de parada hacer
5:     #pragma omp single
6:        idDominados, numDominados, idDominantes ← 0
7:        for i = 1 hasta tamañoPob do
8:           if ∃ P_j: P_j ≻ P_i then /* comprobar ∀P_j */
9:              idDominados[numDominados] ← i
10:             numDominados ← numDominados + 1
11:             idDominantes[i] ← idDominantes[i] ∪ j
12:          end if
13:       end for
14:       auxPob ← P
15:    #pragma omp for schedule (tipoPlanificación)
16:    para i = 1 hasta numDominados hacer
17:       idDom ← idDominados[i]
18:       Procesar Solución (P_idDom, auxPob, idDominantes[idDom], β0, γ, α)
19:       Evaluar Solución Generada (P_idDom)
20:    fin para
21:    #pragma omp single
22:       Actualizar Frente Pareto (Frente, P)
23: fin mientras
```

Algoritmo 3 MOEA/D paralelo generacional

```
1:  Inicializar Estructuras de Datos (P, Frente)
2:  #pragma omp parallel num_threads (numHilos)
3:  Inicializar Población (P, tamañoPob, numHilos)
4:  mientras ! criterio de parada hacer
5:     #pragma omp for schedule (tipoPlanificación, T)
6:     para i = 1 hasta tamañoPob hacer
7:        Selección, cruce y mutación (y, P, T, probCruce, probMutación)
8:        Evaluar Solución Generada (y)
9:        #pragma omp critical
10:          Actualizar Punto Referencia (y, z*)
11:       para cada P_j en la vecindad de P_i hacer
12:          si g_j(y, z*) mejora a g_j(P_j, z*) entonces
13:             Actualizar Subproblema Vecino (P_j, y)
14:          fin si
15:       fin para
16:    fin para
17:    #pragma omp single
18:       Actualizar Vecindades Solapadas (P, T)
19:       Actualizar Frente Pareto (Frente, P)
20: fin mientras
```

las operaciones de cálculo de fitness y actualización de poblaciones pueden verse potencialmente afectadas por dependencias de datos y riesgos de lectura/escritura, por lo que deben ser manejadas por un solo hilo a través de la directiva `#pragma omp single` (líneas 5–7 en el Algoritmo 1). La actualización del frente de Pareto al final de cada generación también se incluye entre las operaciones serie consideradas (líneas 13-14). En cambio, la paralelización de la carga de trabajo se focaliza en el bucle de generación de soluciones hijas, dado que sus iteraciones no muestran dependencias de una solución a otra. Por ello, se emplea la directiva `#pragma omp for` para paralelizar dicho bucle (líneas 8–12), empleando políticas de planificación estáticas o dinámicas (cláusula `schedule`) según las características del problema de optimización abordado.

A.2 Calidad de Solución Exclusivamente

Estos diseños incluyen mecanismos de cálculo de fitness de bajo coste computacional, pensados para determinar rápidamente la calidad de los individuos a cambio de descartar información sobre determinadas propiedades multiobjetivo. Esta categoría también incluye métodos donde el problema multiobjetivo original se descompone en subproblemas escalares, los cuales son optimizados usando la información de calidad de solución de subproblemas vecinos.

El primer MOEA en esta categoría es MO-FA [6]. Este algoritmo realiza comparativas de calidad de solución por pares mediante dominancia Pareto [1], actualizando los individuos dominados P_i mediante la expresión: $P_i.s_k = P_i.s_k + \beta_0 e^{-\gamma\delta_{ij}^2}(P_j.s_k - P_i.s_k) + \alpha(rand[0,1] - \frac{1}{2})$. Aquí, s_k representa la k-ésima variable de decisión de las soluciones procesadas, P_j el individuo que domina a P_i, β_0 un factor de atracción, γ un coeficiente de absorción, δ_{ij} la distancia entre P_i y P_j, α un factor de aleatoriedad y $rand[0,1]$ un número aleatorio en el intervalo [0, 1].

El Algoritmo 2 presenta la implementación paralela de MO-FA, en la cual se distribuye el procesamiento de soluciones dominadas entre los hilos de ejecución. En primer lugar, se pre-procesa la población para determinar el número de individuos dominados (líneas 5–13 en el Algoritmo 2), almacenando sus identificadores así como sus contrapartidas dominantes. Esto permite evitar la inclusión de condiciones *if* para determinar la dominancia Pareto dentro del bucle de procesamiento, el cual es paralelizado mediante `#pragma omp for` (líneas 15–20). Dada la variabilidad del número de soluciones dominadas, resulta adecuado el uso de políticas de planificación dinámicas del bucle paralelo en este algoritmo.

El segundo MOEA considerado es MOEA/D [7], el cual implementa mecanismos de descomposición por la que el i-ésimo individuo de la población representa la mejor solución para el i-ésimo subproblema. Cada uno de dichos subproblemas se optimiza procesando información de sus T subproblemas vecinos más próximos, de tal modo que las soluciones se generan mediante operadores evolutivos aplicados en el ámbito de la vecindad. La nueva solución y es usada para actualizar, en primer lugar, el punto de referencia z^* en la descomposición por Tchebycheff. A continuación, se actualizan los individuos P_j de la vecindad en caso de que y mejore las soluciones actualmente almacenadas según la función escalar g_j considerada.

El diseño paralelo de MOEA/D es mostrado en el Algoritmo 3. A fin de realizar el tratamiento paralelo de subproblemas (líneas 5–16 en el Algoritmo 3), es preciso asegurar que el procesamiento de cada partición de vecinos es llevado a cabo por el mismo hilo para respetar las dependencias de datos del algoritmo. Para ello, se definen bloques de T iteraciones en la cláusula `schedule` de `#pragma omp for` (línea 5). El punto de referencia z^* es compartido globalmente, por lo que su actualización debe realizarse bajo `#pragma omp critical` (líneas 9–10). Al finalizar el tratamiento paralelo de los subproblemas en la generación actual, se introduce una sección `#pragma omp single` para actualizar los vecinos solapados entre particiones y el frente de Pareto (líneas 17–19).

B. Diseños No Generacionales

Los MOEAs paralelos no generacionales tienen por objeto permitir a los mecanismos evolutivos proceder

Algoritmo 4 MOABC no generacional: maestro

```
1:  mientras ! criterio de parada hacer
2:    para i = 0 hasta numHilos-2 hacer
3:      mientras FIFO[i] tenga elementos hacer
4:        Lock (SemáforoFIFO[i])
5:        PobIncons.P_indId ← Pop (FIFO[i])
6:        Unlock (SemáforoFIFO[i])
7:      fin mientras
8:    fin para
9:    si se han recuperado soluciones entonces
10:     Ordenación Rápida No Dominada / Crowding
        (PobIncons.P, tamañoPob)
11:     Calcular Probabilidades Selección (PobIn-
        cons.probArray, PobIncons.P, tamañoPob/2)
12:     Lock (SemáforoPob[i]) /*∀i : i = 0 a numHilos-2*/
13:     Intercambiar Punteros (PobCons, PobIncons)
14:     Unlock (SemáforoPob[i]) /*∀i : i = 0 a numHilos-2*/
15:     Actualizar Frente Pareto (Frente, PobCons.P)
16:   fin si
17: fin mientras
18: Notificar Terminación (i) /*∀i : i = 0 a numHilos-2*/
```

Algoritmo 5 MOABC no generacional: trabajador

```
1:  mientras ! notificación de terminación hacer
2:    Lock (SemáforoPob[IDHilo])
3:    si IDHilo < numHilos/2 entonces
4:      Obtener Obrera (obrera, PobCons.P)
5:    si no
6:      Obtener Observadora (observadora, PobCons.P, Pob-
        Cons.probArray)
7:    fin si
8:    Unlock (SemáforoPob[IDHilo])
9:    si IDHilo < numHilos/2 entonces
10:     Búsqueda Obrera (solución, obrera, probMutación)
11:   si no
12:     Búsqueda Observadora (solución, observadora, prob-
        Mutación)
13:   fin si
14:   si solución.contador > límiteIntentos entonces
15:     Búsqueda Exploradora (solución, probMutación)
16:   fin si
17:   Lock (SemáforoFIFO[IDHilo])
18:   Push (FIFO[IDHilo], solución)
19:   Unlock (SemáforoFIFO[IDHilo])
20: fin mientras
```

tan pronto como una nueva solución candidata haya sido generada. Esto es, la ejecución del algoritmo puede continuar sin esperar a la terminación de todas las tareas paralelas, desapareciendo el concepto tradicional de generación. Para ello, en estos MOEAs se adopta un estilo asíncrono de programación paralela.

Esta categoría es representada por el MOABC [8]. Este método incluye tres mecanismos de búsqueda. El primero de ellos, explotación por abejas obreras, opera sobre la primera mitad de la población, generando nuevas soluciones P_i' a partir de individuos P_i usando la expresión: $P_i'.s_k = P_i.s_k + \phi(P_i.s_k - P_j.s_k)$. Aquí, s_k es la variable de decisión k-ésima, P_j un individuo de la población seleccionado aleatoriamente y ϕ un número aleatorio en el intervalo [-1, 1].

El segundo mecanismo, explotación por abejas observadoras, opera sobre la segunda mitad de la población. Para ello, se calculan probabilidades de selección para cada solución obrera, evaluando su calidad mediante ordenación no dominada y *crowding*. Sobre el individuo seleccionado, se aplican mutaciones para generar nuevas soluciones. Durante estos pasos, se registran los intentos en que una solución no pudo ser mejorada, de manera que, alcanzado un determinado límite, dicha solución es descartada y reemplazada por una aleatoria conforme al tercer mecanismo del algoritmo, la búsqueda por abejas exploradoras.

El diseño paralelo de MOABC organiza los hilos de ejecución, tras su inicialización con `#pragma omp parallel`, bajo roles maestro-trabajador. Dos estructuras poblaciones son necesarias: 1) una estructura consistente para mantener visible el estado actual de la población a los trabajadores y 2) una estructura inconsistente exclusiva del maestro, donde se procesan las soluciones pendientes de integración. Las comunicaciones maestro-trabajador se implementan mediante colas FIFO dedicadas, definiéndose semáforos sobre los accesos concurrentes conflictivos a estructuras de datos compartidas.

El hilo maestro (Algoritmo 4) comprueba la llegada de nuevas soluciones en las colas, almacenándolas en la población inconsistente (líneas 2–8 en el Algoritmo 4). Las nuevas soluciones son integradas en la población tras efectuar ordenaciones y cálculos de probabilidades de selección (líneas 10-11) sobre la estructura inconsistente. Para hacer visible el nuevo estado de la población a los trabajadores, el maestro intercambia los punteros de las estructuras consistente e inconsistente (líneas 12-14).

Las tareas de los trabajadores (Algoritmo 5) comienzan por la lectura de los individuos a procesar de la estructura consistente (líneas 2–8 en el Algoritmo 5), bajo una sección crítica para garantizar que no se produzca durante el cambio de punteros en el lado maestro. Tras ello, el hilo trabajador genera una nueva solución mediante mecanismos de búsqueda obrera u observadora según su identificador (líneas 9–13), aplicando la búsqueda exploradora en caso necesario (líneas 14–16). El resultado es introducido en la cola correspondiente al hilo (líneas 17–19), en exclusión mutua con las lecturas del maestro.

III. Formulación del Problema

Dado que el área de los MOEAs paralelos carece de un framework estándar de benchmarks, el uso de problemas de optimización reales representa un escenario adecuado para la evaluación de estos diseños. En este caso, hemos considerado un problema NP-completo del ámbito de la bioinformática: la reconstrucción filogenética [10]. Este problema tiene por objeto procesar un alineamiento múltiple de secuencias, que contiene información biológica (M nucleótidos o aminoácidos) de N organismos naturales. El objetivo es inferir sus relaciones ancestrales evolutivas, plasmadas en un árbol filogenético $T = (V, E)$ cuya calidad biológica puede ser evaluada mediante distintas funciones objetivo. En este estudio, emplearemos dos funciones filogenéticas: parsimonia $P(T)$ (a minimizar) y verosimilitud $L(T)$ (a maximizar):

$$P(T) = \sum_{i=1}^{M} \sum_{(u,v)\in E} C(u_i, v_i). \tag{1}$$

$$L(T) = \prod_{i=1}^{M} \sum_{x,y\in\Lambda} \pi_x \left[P_{xy}(t_{ru}) L_p\left(u_i = y\right)\right] \times \left[P_{xy}\left(t_{rv}\right) L_p\left(v_i = y\right)\right]. \tag{2}$$

En la Ecuación 1, $(u, v) \in E$ es la rama entre los

TABLA I: Descripción de instancias del problema

Instancia	N	M	Descripción
rbcL_55	55	1314	Nucleótidos del gen rbcL [11]
Fungi_88	88	3329	Proteínas de hongos termófilos [12]
mtDNA_186	186	16608	ADN mitocondrial humano [13]
RDPII_218	218	4182	ARN procariota [14]
ZILLA_500	500	759	Nucleótidos del gen rbcL [15]

TABLA II: Configuración paramétrica de MOEAs (N_C = número de cores)

Criterio de parada		10000 evaluaciones	
SPEA2		MO-FA	
Tamaño población	96	Tamaño población	128
Probabilidad cruce	70 %	Factor atracción β_0	1
Probabilidad mutación	5 %	Coeficiente absorción γ	0,50
Tamaño archivo	75	Factor aleatorio α	0,05
NSGA-II		MOABC	
Tamaño población	96	Tamaño población	96
Probabilidad cruce	70 %	Probabilidad mutación	5 %
Probabilidad mutación	5 %	Límite intentos	25
IBEA		MOEA/D	
Tamaño población	96	Tamaño población	96
Probabilidad cruce	70 %	Probabilidad cruce	70 %
Probabilidad mutación	5 %	Probabilidad mutación	5 %
Factor de escala fitness κ	0,05	Tamaño vecindad T	96 / N_C
Referencia hipervolumen	(2, 2)		

nodos $u, v \in V$, u_i y v_i el estado del carácter i-ésimo de u y v conforme al alfabeto de ADN o aminoácidos Λ y C cuantifica si existe divergencia (1) o no (0) entre u_i y v_i. En el caso de la Ecuación 2, π_x representa la probabilidad estacionaria del estado $x \in \Lambda$, $\mathrm{P}_{xy}(t)$ la probabilidad de mutación desde x a otro estado y en un intervalo temporal t, $r \in V$ el nodo raíz con hijos u, v y $L_p(u_i = y)$, $L_p(v_i = y)$ la verosimilitud parcial de observar y en el carácter i-ésimo de u y v.

La adaptación de los algoritmos al problema involucra una codificación indirecta de soluciones basada en matrices de distancias evolutivas, las cuales son procesadas y mapeadas al espacio de árboles filogenéticos mediante un método de reconstrucción (por ejemplo, *neighbour-joining*). Los operadores evolutivos implementados incluyen, para NSGA-II, SPEA2 e IBEA, selección por torneo binario, cruce uniforme y mutación basada en distribución gamma. MOEA/D emplea los mismos operadores de cruce y mutación, realizando su selección sobre subproblemas aleatorios en la vecindad del actualmente en procesamiento. MO-FA y MOABC adaptan sus ecuaciones al entorno matricial (una variable de decisión = una entrada en la matriz). Finalmente, las implementaciones hacen uso de políticas de planificación dinámica de bucles paralelos, dada la variabilidad en los tiempos de generación y evaluación según las características topológicas de la solución.

IV. Evaluación Experimental

A fin de evaluar los diferentes componentes taxonómicos considerados, se han realizado experimentos en distintas infraestructuras hardware de memoria compartida. Particularmente, se han empleado:

1. Sistema HPC de 48 cores compuesto por 4 procesadores AMD Opteron 6174 a 2,2GHz, 12MB de caché L3 y 64GB de RAM DDR3.
2. Sistema medio de 16 cores compuesto por dos procesadores Intel Xeon E5-2630v3 a 2,4GHz, 20MB de caché L3 y 80GB de RAM DDR3.
3. Sistema comercial de 4 cores con CPU Intel i7-2600 a 3,4GHz, 8MB de caché L3 y 8GB de RAM DDR3.

TABLA III: Tiempos de ejecución de las versiones serie (en segundos, sistema AMD)

Instancia	NSGA-II	SPEA2	IBEA
rbcL_55	5294,76	5355,62	5372,39
Fungi_88	54202,84	54602,34	54982,11
mtDNA_186	47135,49	47313,75	48083,64
RDPII_218	50452,11	51045,57	51334,49
ZILLA_500	69702,12	70377,89	71307,04
Instancia	MO-FA	MOEA/D	MOABC
rbcL_55	5486,04	5247,58	5610,64
Fungi_88	56424,79	53930,55	52370,39
mtDNA_186	47700,12	47116,05	45379,13
RDPII_218	54507,41	49104,02	48218,18
ZILLA_500	77365,74	68802,96	65459,84

TABLA IV: Tiempos de ejecución de las versiones serie (en segundos, sistemas Intel)

	NSGA-II		SPEA2		IBEA	
Instancia	Xeon	i7	Xeon	i7	Xeon	i7
rbcL_55	2728,15	2501,44	2784,75	2520,71	2793,45	2528,84
Fungi_88	29854,38	29387,12	30225,22	29782,92	30417,72	30037,95
mtDNA_186	24344,49	23455,70	24468,90	23796,92	24569,57	23791,62
RDPII_218	25850,61	24254,79	25916,94	24477,56	26360,49	24614,88
ZILLA_500	35537,28	33214,71	35625,95	33305,39	35775,17	33379,55
	MO-FA		MOEA/D		MOABC	
Instancia	Xeon	i7	Xeon	i7	Xeon	i7
rbcL_55	2798,48	2538,53	2697,85	2470,54	2810,04	2586,45
Fungi_88	31442,41	30652,01	29732,70	28652,08	29737,44	28244,67
mtDNA_186	24624,69	24236,39	24089,69	23124,18	23349,16	22203,46
RDPII_218	26522,65	24580,83	25497,85	24100,44	25045,55	23689,02
ZILLA_500	36078,48	33780,20	35209,83	33035,28	35127,26	32793,54

La experimentación se ha efectuado sobre cinco conjuntos biológicos reales descritos en la Tabla I, los cuales proporcionan una muestra representativa de tamaños del problema en función del número de secuencias y longitud de secuencia. A su vez, la Tabla II muestra la configuración óptima de parámetros de entrada encontrada para cada MOEA, conforme a los estudios paramétricos explicados en [16].

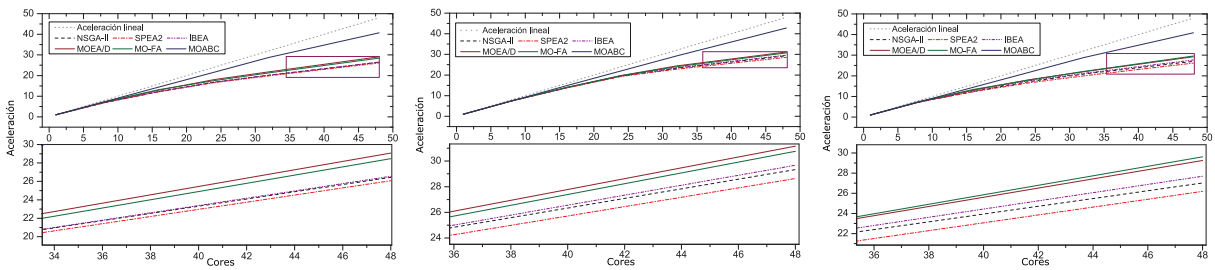
A. Métricas y Resultados

Las métricas consideradas en la evaluación engloban indicadores de rendimiento paralelo y multiobjetivo. En el caso del rendimiento paralelo, se han evaluado las ganancias temporales obtenidas mediante las métricas de aceleración y eficiencia [9]. Estas métricas permiten medir la mejora en tiempo de ejecución observada con respecto a la versión serie de la aplicación, así como la utilización media de unidades de procesamiento. Por su parte, la calidad multiobjetivo se ha analizado en base a las métricas de hipervolumen y espaciado [1]. El hipervolumen calcula el área del espacio objetivo que es cubierto por al menos un punto del frente generado por el MOEA evaluado. El espaciado, por su parte, mide la uniformidad de la distribución del frente de Pareto, conforme a las distancias observadas entre puntos vecinos.

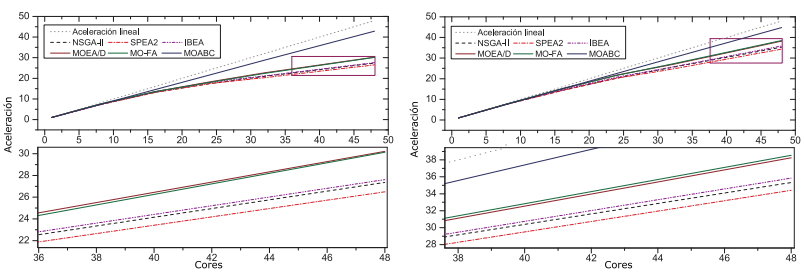
En primer lugar nos centraremos en la evaluación de rendimiento paralelo. Tomando como referencia los tiempos serie de cada MOEA (Tablas III y IV), se han obtenido las aceleraciones y eficiencias medianas de 11 ejecuciones independientes para distintos tamaños de sistema. La Tabla V presenta los resulta-

TABLA V: Resultados de rendimiento paralelo: aceleraciones y eficiencias (sistema AMD Opteron)

	Aceleración					Eficiencia(%)				
Algoritmo	8 cores	16 cores	24 cores	32 cores	48 cores	8 cores	16 cores	24 cores	32 cores	48 cores
rbcL_55										
NSGA-II	6,831±0,087	12,279±0,172	16,772±0,178	20,178±0,284	26,435±0,444	85,393	76,745	69,884	63,057	55,073
SPEA2	6,873±0,067	12,157±0,168	16,553±0,159	19,865±0,286	26,075±0,405	85,912	75,978	68,970	62,077	54,323
IBEA	6,950±0,104	12,321±0,137	16,834±0,232	20,208±0,233	26,556±0,497	86,869	77,005	70,142	63,149	55,326
MO-FA	7,206±0,079	13,217±0,099	17,574±0,238	21,332±0,254	28,471±0,416	90,079	82,607	73,226	66,661	59,315
MOEA/D	7,073±0,054	12,949±0,140	18,055±0,275	21,824±0,313	29,074±0,436	88,408	80,936	75,229	68,200	60,571
MOABC	**7,235±0,072**	**14,464±0,167**	**21,639±0,171**	**28,823±0,250**	**40,694±0,322**	**90,432**	**90,398**	**90,161**	**90,072**	**84,779**
Fungi_88										
NSGA-II	7,406±0,040	13,849±0,127	19,533±0,206	23,338±0,372	29,333±0,515	92,576	86,553	81,386	72,931	61,110
SPEA2	7,403±0,015	13,712±0,130	19,327±0,245	22,796±0,380	28,634±0,510	92,540	85,703	80,529	71,237	59,654
IBEA	7,405±0,036	13,869±0,075	19,555±0,181	23,458±0,458	29,665±0,483	92,559	86,679	81,477	73,307	61,802
MO-FA	**7,490±0,024**	13,934±0,138	19,764±0,244	24,029±0,310	30,765±0,533	**93,620**	87,086	82,348	75,092	64,093
MOEA/D	7,484±0,054	13,956±0,140	19,788±0,276	24,406±0,313	31,159±0,436	93,552	87,228	82,448	76,268	64,914
MOABC	7,424±0,041	**14,810±0,122**	**22,037±0,233**	**29,219±0,312**	**42,808±0,470**	92,794	**92,562**	**91,820**	**91,309**	**89,184**
mtDNA_186										
NSGA-II	7,208±0,148	12,405±0,264	17,084±0,359	20,858±0,405	27,018±0,518	90,096	77,533	71,184	65,181	56,288
SPEA2	7,170±0,143	12,346±0,260	16,542±0,347	19,950±0,426	26,188±0,508	89,621	77,164	68,924	62,344	54,558
IBEA	7,210±0,144	12,897±0,271	17,562±0,278	21,173±0,392	27,695±0,386	90,122	80,604	73,173	66,165	57,698
MO-FA	**7,453±0,144**	13,402±0,251	17,880±0,409	22,077±0,421	29,620±0,527	**93,161**	83,763	74,500	68,990	61,709
MOEA/D	7,305±0,062	13,114±0,132	18,059±0,205	21,953±0,389	29,248±0,345	91,311	81,963	75,247	68,602	60,934
MOABC	7,243±0,058	**14,455±0,167**	**21,636±0,153**	**28,835±0,202**	**40,850±0,316**	90,537	**90,341**	**90,151**	**90,109**	**85,104**
RDPII_218										
NSGA-II	7,277±0,158	13,177±0,252	17,425±0,373	20,943±0,429	27,358±0,591	90,960	82,355	72,604	65,447	56,996
SPEA2	7,153±0,123	13,093±0,241	17,490±0,374	20,348±0,424	26,495±0,571	89,418	81,832	72,875	63,586	55,197
IBEA	7,297±0,135	13,369±0,261	18,007±0,393	21,211±0,574	27,617±0,759	91,207	83,558	75,031	66,283	57,535
MO-FA	**7,509±0,121**	13,576±0,212	18,062±0,416	22,376±0,462	30,134±0,626	**93,866**	84,849	75,260	69,925	62,779
MOEA/D	7,309±0,171	13,465±0,226	18,247±0,417	22,673±0,587	30,217±0,860	91,366	84,153	76,027	70,854	62,951
MOABC	7,260±0,084	**14,501±0,178**	**21,627±0,209**	**28,812±0,255**	**42,864±0,362**	90,745	**90,632**	**90,111**	**90,037**	**89,301**
ZILLA_500										
NSGA-II	7,650±0,089	14,260±0,177	20,254±0,226	25,459±0,383	35,325±0,421	95,622	89,127	84,391	79,561	73,593
SPEA2	7,642±0,155	14,289±0,192	20,222±0,271	24,597±0,373	34,400±0,479	95,526	89,306	84,257	76,864	71,668
IBEA	7,684±0,181	14,572±0,103	20,726±0,199	25,637±0,311	35,854±0,505	96,054	91,075	86,358	80,115	74,696
MO-FA	**7,764±0,087**	**15,113±0,253**	21,589±0,359	27,121±0,362	38,536±0,548	**97,050**	**94,455**	89,952	84,754	80,284
MOEA/D	7,702±0,095	15,066±0,210	21,738±0,125	26,845±0,379	38,240±0,727	96,277	94,164	90,574	83,890	79,668
MOABC	7,532±0,096	15,052±0,162	**22,568±0,243**	**30,060±0,356**	**44,761±0,424**	94,156	94,072	**94,034**	**93,937**	**93,252**
Resultados promedio										
NSGA-II	7,274	13,194	18,214	22,155	29,094	90,930	82,463	75,890	69,235	60,612
SPEA2	7,248	13,119	18,027	21,511	28,358	90,603	81,996	75,112	67,223	59,080
IBEA	7,309	13,406	18,537	22,337	29,478	91,364	83,784	77,236	69,804	61,412
MO-FA	**7,484**	13,848	18,974	23,387	31,505	**93,555**	86,553	79,058	73,084	65,636
MOEA/D	7,375	13,710	19,177	23,540	31,588	92,183	85,689	79,905	73,563	65,808
MOABC	7,339	**14,656**	**21,901**	**29,150**	**42,395**	91,735	**91,603**	**91,256**	**91,093**	**88,324**



(a) rbcL_55 (b) Fungi_88 (c) mtDNA_186



(d) RDPII_218 (e) ZILLA_500

Fig. 3: Evolución de aceleraciones (términos medianos para el sistema de 48 cores AMD Opteron)

dos observados para el caso del sistema AMD en configuraciones de 8, 16, 24, 32 y 48 cores. En términos generales, el diseño no generacional de MOABC proporciona la mejor escalabilidad paralela en el sistema HPC, acelerando los cómputos del algoritmo hasta 44,76x al emplear 48 cores. En el lado generacional, MO-FA y MOEA/D (calidad de solución exclusivamente) obtienen los resultados más satisfactorios e incluso mejoran a MOABC cuando el número de cores usados es bajo. En cambio, NSGA-II, SPEA2 e IBEA reportan los resultados paralelos más discretos al acrecentarse el impacto de las sincronizaciones y los elementos serie de esta tendencia algorítmica (con medidas de calidad de solución y diversidad). La Fi-

TABLA VI: Resultados de rendimiento paralelo: aceleraciones y eficiencias (sistema Intel Xeon)

	Aceleración		Eficiencia(%)	
Algoritmo	8 cores	16 cores	8 cores	16 cores
		rbcL_55		
NSGA-II	7,293±0,068	12,219±0,233	91,164	76,370
SPEA2	7,201±0,095	12,182±0,235	90,009	76,140
IBEA	7,352±0,087	12,250±0,092	91,897	76,560
MO-FA	**7,424±0,057**	12,907±0,072	**92,803**	80,669
MOEA/D	7,364±0,093	12,628±0,164	92,056	78,926
MOABC	7,305±0,062	**13,753±0,067**	91,316	**85,955**
		Fungi_88		
NSGA-II	7,337±0,073	13,762±0,077	91,711	86,015
SPEA2	7,216±0,012	13,541±0,148	90,196	84,633
IBEA	7,470±0,050	13,974±0,183	93,370	87,337
MO-FA	7,523±0,059	14,131±0,073	94,039	88,319
MOEA/D	**7,626±0,010**	**14,310±0,024**	**95,322**	**89,441**
MOABC	7,481±0,057	14,237±0,061	93,513	88,980
		mtDNA_186		
NSGA-II	7,331±0,119	12,471±0,201	91,633	77,942
SPEA2	7,235±0,131	12,461±0,272	90,436	77,880
IBEA	7,434±0,132	12,514±0,202	92,927	78,214
MO-FA	**7,579±0,146**	12,950±0,225	**94,733**	80,939
MOEA/D	7,541±0,130	12,875±0,166	94,269	80,466
MOABC	7,529±0,019	**14,273±0,050**	94,110	**89,203**
		RDPII_218		
NSGA-II	7,323±0,154	13,114±0,158	91,535	81,961
SPEA2	7,279±0,101	13,021±0,338	90,990	81,381
IBEA	7,397±0,135	13,242±0,365	92,456	82,763
MO-FA	**7,538±0,197**	13,844±0,384	**94,219**	86,523
MOEA/D	7,478±0,090	13,758±0,457	93,478	85,985
MOABC	7,395±0,106	**14,382±0,200**	92,442	**89,890**
		ZILLA_500		
NSGA-II	7,403±0,074	13,955±0,281	92,532	87,220
SPEA2	7,410±0,020	13,850±0,191	92,621	86,564
IBEA	7,399±0,121	14,184±0,282	92,491	88,649
MO-FA	7,637±0,106	**14,527±0,319**	95,460	**90,794**
MOEA/D	**7,646±0,038**	14,494±0,183	**95,580**	90,585
MOABC	7,353±0,124	14,429±0,309	91,910	90,181
		Resultados promedio		
NSGA-II	7,337	13,104	91,715	81,902
SPEA2	7,268	13,011	90,850	81,320
IBEA	7,410	13,233	92,628	82,705
MO-FA	**7,540**	13,672	**94,251**	85,449
MOEA/D	7,531	13,613	94,141	85,081
MOABC	7,413	**14,215**	92,658	**88,842**

TABLA VII: Resultados de rendimiento paralelo: aceleraciones y eficiencias (sistema Intel i7)

	Aceleración	Eficiencia(%)	Aceleración	Eficiencia(%)
Algoritmo	4 cores	4 cores	4 cores	4 cores
	rbcL_55		Fungi_88	
NSGA-II	3,677±0,052	91,920	3,860±0,041	96,495
SPEA2	3,556±0,044	88,907	3,853±0,021	96,317
IBEA	3,698±0,018	92,458	3,877±0,020	96,913
MO-FA	**3,739±0,020**	**93,464**	3,882±0,017	97,055
MOEA/D	3,724±0,043	93,099	**3,893±0,042**	**97,323**
MOABC	3,402±0,045	85,051	3,558±0,030	88,958
	mtDNA_186		RDPII_218	
NSGA-II	3,729±0,019	93,225	3,825±0,083	95,632
SPEA2	3,712±0,017	92,811	3,821±0,066	95,535
IBEA	3,771±0,055	94,279	3,831±0,060	95,774
MO-FA	**3,797±0,039**	**94,930**	**3,879±0,017**	**96,971**
MOEA/D	3,796±0,012	94,909	3,855±0,026	96,386
MOABC	3,556±0,013	88,911	3,551±0,016	88,773
	ZILLA_500		Resultados promedio	
NSGA-II	3,863±0,026	96,568	3,791	94,768
SPEA2	3,862±0,032	96,562	3,761	94,026
IBEA	3,895±0,015	97,380	3,814	95,361
MO-FA	**3,949±0,038**	**98,724**	**3,849**	**96,229**
MOEA/D	3,946±0,032	98,657	3,843	96,075
MOABC	3,560±0,033	89,012	3,526	88,141

gura 3 representa las aceleraciones conseguidas por cada diseño, pudiéndose distinguir las tres tendencias fundamentales consideradas en la taxonomía.

La Tabla VI presenta los resultados paralelos observados en el sistema medio Intel Xeon, considerando configuraciones de 8 y 16 cores, mientras que los resultados del sistema comercial Intel i7 de 4 cores vienen dados en la Tabla VII. Los resultados para 16 cores en la Tabla VI dan cuenta del hecho de que el diseño no generacional de MOABC permite obtener aceleraciones medias de 14,22x, representando pues la mejor tendencia en términos generales con una eficiencia media del 88,84 %. Sin embargo, la experimentación también sugiere que las aproximaciones generacionales basadas en medidas de calidad de solución (MO-FA y MOEA/D) consiguen los mejores resultados en las instancias más costosas a nivel temporal, Fungi_88 y ZILLA_500. Es más, estos diseños algorítmicos representan la mejor tendencia en configuraciones de 8 cores. La tendencia generacional representada por NSGA-II, SPEA2 e IBEA también obtiene resultados satisfactorios en este escenario, especialmente en el caso de ZILLA_500 donde mejoran los resultados del método no generacional.

Con respecto a la Tabla VII, los resultados para 4 cores en la plataforma comercial i7 apuntan a la tendencia generacional representada por MO-FA y MOEA/D como la que consigue la mejor capacidad de explotación de recursos, con eficiencias de hasta 98,72 % / 98,66 %. Los diseños generacionales basados en calidad de solución y diversidad se alzan como la segunda mejor aproximación en este tipo de infraestructuras, con eficiencias medias por encima del 94 %. En cambio, la categoría no generacional representada por MOABC obtiene eficiencias en el rango 85,05 % – 89,01 %, no alcanzando el rendimiento paralelo de las categorías generacionales. Estos resultados dan cuentan de la problemática del «core maestro» en sistemas de este estilo, al exigir el diseño no generacional la dedicación de un core completo a tareas de gestión y actualización de la población sin intervenir en las tareas paralelas del algoritmo.

La segunda perspectiva de análisis considerada en este estudio es la de rendimiento multiobjetivo. Con objeto de examinar la calidad de cada una de las tendencias bajo estudio, hemos analizado mediante hipervolumen y espaciado los frentes de Pareto obtenidos en 31 ejecuciones independientes por experimento. La Tabla VIII proporciona los resultados medianos de hipervolumen observados en la experimentación (parte superior de la tabla), así como los valores de espaciado para los frentes medianos (parte inferior). Los valores de hipervolumen dan cuenta de la bondad del MOABC no generacional, representando el algoritmo que consigue los resultados más notables desde la perspectiva de esta métrica. Concretamente, MOABC obtiene un hipervolumen medio del 73,5 %, consiguiendo el mejor resultado en rbcL_55, Fungi_88, mtDNA_186 y ZILLA_500. Los siguientes algoritmos apuntados por la métrica de hipervolumen son los diseños generacionales de MO-FA (73,4 %), el cual consigue el mejor resultado en RDPII_218, y MOEA/D (73,3 %). NSGA-II, SPEA2 e IBEA obtienen resultados competitivos a su vez, particularmente en el caso de IBEA conforme a los resultados obtenidos en instancias del problema con un número de secuencias bajo (rbcL_55 y Fungi_88).

No obstante, estos diseños generacionales que incluyen medidas de diversidad juegan un papel más relevante en la comparativa de espaciado, al dar lugar a distribuciones de soluciones satisfactorias en

TABLA VIII: Resultados de rendimiento multiobjetivo: hipervolumen y espaciado

Instancia	NSGA-II	SPEA2	IBEA	MO-FA	MOEA/D	MOABC
			Hipervolumen (%)			
rbcL_55	71,237±0,155	71,227±0,153	71,432±0,052	71,473±0,079	71,418±0,048	**71,732±0,018**
Fungi_88	77,797±0,021	77,790±0,047	77,809±0,015	77,814±0,044	77,806±0,028	**77,872±0,019**
mtDNA_186	69,718±0,112	69,631±0,158	69,830±0,084	70,004±0,008	69,981±0,006	**70,021±0,014**
RDPII_218	73,625±0,054	73,727±0,066	74,302±0,062	**74,725±0,076**	74,631±0,118	74,641±0,050
ZILLA_500	71,800±0,034	71,799±0,019	72,337±0,042	72,959±0,023	72,671±0,026	**73,016±0,012**
Promedio	72,835	72,834	73,142	73,395	73,301	**73,456**
			Espaciado (%)			
rbcL_55	0,080	**0,060**	0,094	0,128	0,102	0,087
Fungi_88	0,078	**0,059**	0,077	0,123	0,083	0,067
mtDNA_186	0,101	0,099	**0,078**	0,103	0,107	0,079
RDPII_218	0,025	0,043	0,021	0,033	0,029	**0,017**
ZILLA_500	0,111	0,127	0,108	0,051	0,091	**0,031**
Promedio	0,079	0,077	0,076	0,087	0,082	**0,056**

tres de los conjuntos de datos analizados. Todo ello da lugar a valores medios de espaciado por debajo del 0,08, mejorando los resultados reportados por los diseños basados en calidad de solución exclusivamente. Es posible verificar un rendimiento significativo en diversidad también para el caso del método no generacional MOABC, al permitir integrar tanto calidad de solución como densidad en las estrategias de gestión poblacional del lado maestro. Como resultado, se obtiene un rendimiento notable en RDPII_218 y ZILLA_500, con un valor medio de 0,06.

B. Implicaciones

Nuestra evaluación experimental da cuenta de las principales oportunidades y desventajas de cada diseño MOEA considerado en la taxonomía intra-algorítmica. Al acometer la elección entre distintas aproximaciones paralelas, debemos tener en cuenta no solo las preferencias del experto en términos multiobjetivo, sino también las características de la plataforma hardware donde se ejecutarán los MOEAs.

El primer escenario a considerar es aquel en el que el hardware disponible para la ejecución del MOEA paralelo es de tipo medio o comercial. Los sistemas accesibles a nivel comercial se caracterizan por la inclusión de un único procesador multicore o por un número limitado de ellos, con un total de entre 4 y 8 cores de procesamiento (aunque con tendencia progresiva hacia números mayores, como por ejemplo 16 cores). De acuerdo a nuestros resultados, las aproximaciones de tipo generacional representan una elección adecuada para alcanzar una explotación precisa de recursos paralelos en estos sistemas. En el caso de poner prioridad a la obtención de eficiencias paralelas altas, es posible confiar en diseños basados en calidad de solución exclusivamente, como MO-FA y MOEA/D. De hecho, esta categoría generacional presenta la ventaja adicional de obtener un rendimiento paralelo significativo también en infraestructuras de tamaño medio, con relevantes resultados multiobjetivo desde la perspectiva del hipervolumen. Por otra parte, la tendencia generacional basada en calidad de solución y diversidad (NSGA-II, SPEA2 e IBEA) puede ser seleccionada en caso de que se desee obtener frentes de Pareto con mejor distribución de soluciones. Finalmente, la aproximación no generacional puede conllevar una degradación de rendimiento especialmente en infraestructuras con recursos paralelos limitados, tal y como muestran los resultados obtenidos en la configuración de 4 cores del Intel i7. Particularmente, la solución de problemas multiobjetivo de elevado coste computacional se ve afectada por la dedicación exclusiva de tareas de soporte en el core maestro, sin poder colaborar en la paralelización de tareas que dominan el tiempo de ejecución.

El segundo escenario es aquel en el que se encuentran habilitados sistemas de computación de altas prestaciones para la ejecución del MOEA paralelo. Las configuraciones hardware HPC incluyen un número elevado de unidades de procesamiento cuya explotación depende de las capacidades de escalabilidad del diseño paralelo considerado. Es en este escenario donde la aproximación no generacional representa la opción más adecuada, al permitir la explotación precisa del potencial de procesamiento paralelo de estos sistemas. No obstante, diseños generacionales como MO-FA y MOEA/D podrían aplicarse en el caso de instancias de problema complejas, como ocurre en nuestro caso para ZILLA_500. Una característica particular de los MOEAs paralelos a considerar en este contexto es la limitación teórica en escalabilidad impuesta por el tamaño de la población. En contextos multiobjetivo, es habitual encontrarse con procesos de optimización complejos que requieren definir poblaciones con un número de individuos grande, siendo normalmente superior al número de cores disponibles en los sistemas multiprocesadores actuales. En este caso, el límite teórico no provoca un impacto en el rendimiento paralelo. Sin embargo, dicho límite resulta clave en la explotación de sistemas HPC con miles de cores de procesamiento (superando el número de individuos = tareas a procesar en paralelo). Para poder lidiar con la limitación impuesta por el tamaño de la población, es posible aplicar estrategias de mayor granularidad (paralelismo a nivel de tarea para ejecutar simultáneamente múltiples ejecuciones independientes del MOEA paralelo) o bien establecer cambios en el diseño algorítmico para favorecer su ejecución en entornos HPC. Por ejemplo, múltiples hilos pueden procesar copias del mismo individuo de la población pero aplicando distintos operadores evolutivos, de tal manera que se generen soluciones candidatas diferentes a partir de un punto de referencia común. Sin embargo, es nece-

sario llevar a cabo estudios profundos para poder integrar de manera óptima dichas estrategias, al implicar una modificación en el comportamiento original del MOEA. De este modo, sería posible establecer el umbral entre paralelismo e impacto en la calidad de soluciones más adecuado para el problema abordado.

En resumen, los diseños intra-algoritmo no generacionales representan una opción robusta cuando la complejidad del problema require el uso de configuraciones HPC avanzadas. Por su parte, las aproximaciones generacionales encuentran su ámbito de aplicación en sistemas comerciales y, en general, cuando el acceso a recursos HPC no es posible. En este caso, la selección de una tendencia generacional concreta dependerá de los efectos del diseño algorítmico sobre el rendimiento multiobjetivo y paralelo deseado.

V. Conclusiones

Este trabajo se ha focalizado en el análisis de diseños evolutivos paralelos para optimización multiobjetivo. Dado el continuo crecimiento en complejidad de los problemas multiobjetivo reales, es necesario llevar a cabo estudios rigurosos y detallados para determinar las estrategias más precisas para obtener soluciones de gran calidad en tiempos reducidos. Bajo este supuesto, hemos abordado un análisis comparativo de distintos componentes taxonómicos de la rama de MOEAs paralelos intra-algoritmo, a fin de identificar la influencia de sus características algorítmicas y propiedades paralelas en el rendimiento.

Nuestra experimentación se ha centrado en tres sistemas paralelos de memoria compartida (Intel y AMD). El rendimiento paralelo ha sido evaluado mediante las métricas de aceleración y eficiencia, mientras que las propiedades multiobjetivo de cada MOEA han sido examinadas mediante el hipervolumen y el espaciado. Dichas métricas han dado cuenta de las ventajas e inconvenientes de cada tendencia. En términos de escalabilidad paralela, se ha identificado el diseño no generacional como la técnica más adecuada para explotar infraestructuras HPC con números de cores elevados. Seguidamente encontramos los diseños generacionales basados en calidad de solución exclusivamente, los cuales consiguen resultados más relevantes en sistemas de tamaño medio y comerciales. La alternativa generacional con mecanismos de calidad y diversidad resulta una elección adecuada también en dichos sistemas cuando se debe poner prioridad a cuestiones multiobjetivo adicionales (p.e. uniformidad). En base a estos resultados, hemos proporcionado una serie de recomendaciones para afrontar la selección de MOEAs paralelos según las características del hardware subyacente.

Como trabajo futuro, se propone extender la evaluación de MOEAs paralelos en otras plataformas hardware, más allá de escenarios de memoria compartida. Afrontaremos la evaluación de diseños paralelos intra-algoritmo en sistemas heterogéneos, entornos distribuidos y configuraciones híbridas de memoria compartida + distribuida. A su vez, se estudiarán alternativas en las que se usen co-procesadores para complementar a los esquemas intra-algoritmo CPU en el procesamiento de operaciones con paralelismo de datos y dependientes del problema. Finalmente, otra línea de investigación radica en identificar los umbrales del paralelismo no generacional.

Agradecimientos

Este trabajo ha sido parcialmente financiado por la AEI (Agencia Estatal de Investigación, España) y el FEDER (Fondo Europeo de Desarrollo Regional, Unión Europea), bajo el proyecto TIN2016-76259-P (proyecto PROTEIN), así como por la FCT (Fundação para a Ciência e a Tecnologia, Portugal) con referencia UID/CEC/50021/2019. Gracias también a la Junta de Extremadura y el FEDER por la ayuda GR18090 otorgada al grupo de investigación TIC015. Sergio Santander-Jiménez agradece a la FCT su contrato postdoctoral SFRH/BPD/119220/2016.

Referencias

[1] C. Coello, C. Dhaenens, and L. Jourdan, *Advances in Multi-Objective Nature Inspired Computing*, Springer Verlag, Berlin / Heidelberg, 2010.

[2] E. G. Talbi, "Parallel Evolutionary Combinatorial Optimization," in *Springer Handbook of Computational Intelligence*, pp. 1107–1125. Springer, 2015.

[3] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A Fast and Elitist Multi–Objective Genetic Algorithm: NSGA-II," *IEEE Trans. Evol. Comput.*, vol. 6, no. 2, pp. 182–197, 2002.

[4] E. Zitzler, M. Laumanns, and L. Thiele, "SPEA2: Improving the Strength Pareto Evolutionary Algorithm," in *Proc. of EUROGEN'02*, 2002, pp. 95–100.

[5] E. Zitzler and S. Künzli, "Indicator-Based Selection in Multiobjective Search," in *PPSN VIII*. 2004, vol. 3242 of *LNCS*, pp. 832–842, Springer Verlag.

[6] X. S. Yang, "Firefly algorithm, stochastic test functions and design optimisation," *International Journal of Bio-Inspired Computation*, vol. 2, no. 2, pp. 78–84, 2010.

[7] Q. Zhang and H. Li, "MOEA/D: A Multi-objective Evolutionary Algorithm Based on Decomposition," *IEEE Trans. Evol. Comput.*, vol. 11, no. 6, pp. 712–731, 2007.

[8] D. Karaboga, B. Gorkemli, C. Ozturk, and N. Karaboga, "A comprehensive survey: artificial bee colony (ABC) algorithm and applications," *Artif. Intell. Rev.*, vol. 42, no. 1, pp. 21–57, 2014.

[9] B. Chapman, G. Jost, and R. van der Pas, *Using OpenMP: Portable Shared Memory Parallel Programming*, The MIT Press, Cambridge, MA, USA, 2007.

[10] M. Steel, *Phylogeny: Discrete and Random Processes in Evolution*, Society for Industrial and Applied Mathematics, Philadelphia, 2016.

[11] P. O. Lewis, "A Genetic Algorithm for Maximum-Likelihood Phylogeny Inference Using Nucleotide Sequence Data," *Molecular Biology and Evolution*, vol. 15, no. 3, pp. 277–283, 1998.

[12] I. Morgenstern et al., "A molecular phylogeny of thermophilic fungi," *Fungal Biology*, vol. 116, no. 4, pp. 489–502, 2012.

[13] M. Ingman and U. Gyllensten, "mtDB: Human Mitochondrial Genome Database, a resource for population genetics and medical sciences," *Nucleic Acids Research*, vol. 34 (Database issue D749-D751), 2006.

[14] J. R. Cole et al., "The Ribosomal Database Project (RDP-II): sequences and tools for high-throughput rRNA analysis," *Nucleic Acids Research*, vol. 33 (Database issue D294-D296), 2005.

[15] M. W. Chase et al., "Phylogenetics of seed plants: An analysis of nucleotide sequences from the plastid gene rbcl," *Annals of the Missouri Botanical Garden*, vol. 80, no. 3, pp. 528–580, 1993.

[16] S. Santander-Jiménez and M. A. Vega-Rodríguez, "Comparative Analysis of Intra-Algorithm Parallel Multiobjective Evolutionary Algorithms: Taxonomy Implications on Bioinformatics Scenarios," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 1, pp. 63–78, 2019.

Análisis Comparativo de Tecnologías GPGPU para Acelerar Funciones Objetivo: Parsimonia como Caso de Estudio

Sergio Santander-Jiménez[1], Miguel A. Vega-Rodríguez[2], Jorge Vicente-Viola[2] y Leonel Sousa[1]

Resumen— **La complejidad temporal de los métodos de optimización actuales se encuentra fundamentalmente gobernada por los cálculos asociados a las funciones objetivo, las cuales proporcionan medidas fehacientes para evaluar la calidad de las soluciones. Por este motivo, gran parte de los esfuerzos investigadores se han focalizado en la propuesta de implementaciones eficientes que aprovechen las oportunidades de paralelismo presentes en estas funciones. Este trabajo explora tecnologías GPGPU para acelerar funciones objetivo, considerando como caso de estudio la paralelización de la función filogenética de parsimonia. Particularmente, abordamos la evaluación comparativa de distintos modelos de programación y arquitecturas GPU, destacando las ventajas e inconvenientes de cada aproximación. Los experimentos realizados dan cuenta de la relación existente entre las características de los datos de entrada y la utilización efectiva de los recursos GPU, poniéndose de manifiesto la relevancia de estas arquitecturas en la resolución de escenarios de optimización actuales y futuros.**

Palabras clave— **Algoritmos Paralelos, GPGPU, Funciones Objetivo, Bioinformática.**

I. Introducción

La explotación del paralelismo para acelerar la resolución de problemas de optimización se ha convertido en una cuestión fundamental en el ámbito investigador. Los métodos de optimización actuales, como es el caso de las metaheurísticas, incluyen en sus cálculos el cómputo de una o varias funciones objetivo $f : S \rightarrow \Re$ (donde S representa el espacio de búsqueda) para evaluar de manera precisa la calidad de las soluciones candidatas. La idea de minimizar los costes de estas funciones representa una cuestión clave en escenarios de optimización reales, al contribuir con porcentajes dominantes en los tiempos de ejecución globales. Este es el caso de las funciones objetivo bioinformáticas, cuya naturaleza costosa viene dada por el elevado número de operaciones complejas a realizar sobre las secuencias biológicas. Así, la tarea de evaluar soluciones en conjuntos de datos reales implica el procesamiento de miles de nucleótidos o aminoácidos, provocando un impacto temporal significativo. En este sentido, los cálculos de las funciones objetivo a menudo se basan en el concepto de repetir el mismo conjunto de operaciones sobre múltiples elementos de datos de manera independiente, mostrando oportunidades inherentes de paralelismo.

El empleo de aceleradores hardware y, especialmente, GPUs ha tomado un rol predominante en la mejora de aplicaciones con paralelismo de datos. De hecho, las aproximaciones GPGPU han dado cuenta de resultados significativos en la resolución de problemas bioinformáticos como el alineamiento de secuencias y el acoplamiento molecular [1]. No obstante, es preciso acometer trabajos adicionales en esta dirección para afrontar dos cuestiones clave. En primer lugar, es necesario proporcionar una caracterización rigurosa de los kernels bioinformáticos para comprender la relación entre el tamaño del problema y la explotación efectiva de recursos hardware. Es más, dicha caracterización puede arrojar luz sobre la utilidad de las GPUs sobre otras arquitecturas paralelas, a fin de poder predecir la plataforma de cómputo más adecuada en escenarios de optimización futuros. En segundo lugar, la disponibilidad de distintas librerías de programación y dispositivos GPU exige realizar estudios comparativos sistemáticos para cuantificar las ventajas e inconvenientes de cada aproximación, en términos de rendimiento y productividad.

Este trabajo aborda la evaluación comparativa de tecnologías GPGPU para paralelizar funciones objetivo. Como caso de estudio, nos centraremos en la aceleración de la función filogenética de parsimonia [2], la cual ha sido objeto de investigaciones previas en plataformas CPU y FPGAs [3] así como en sistemas heterogéneos [4]. Este estudio investiga implementaciones del kernel de parsimonia mediante tres frameworks de programación GPU: CUDA [5], OpenCL [6] y OpenACC [7]. La evaluación de las aproximaciones propuestas será efectuada en distintos dispositivos GPU a fin de destacar las ventajas e inconvenientes de las técnicas y arquitecturas estudiadas conforme a las características del kernel. Además, examinaremos la relevancia de los diseños GPU con respecto a otras arquitecturas paralelas (sistemas CPU multiprocesador e Intel Xeon Phi), así como otras implementaciones del estado del arte.

Este artículo está organizado del siguiente modo. La siguiente sección proporciona la formulación de la función filogenética de parsimonia, mientras que la Sección III presenta los diseños GPU propuestos. La Sección IV explica la metodología experimental considerada y discute los resultados obtenidos. Finalmente, la Sección V presenta las conclusiones de este estudio, destacando líneas de trabajo futuro.

[1]Instituto de Engenharia de Sistemas e Computadores - Investigação e Desenvolvimento em Lisboa (INESC-ID), Instituto Superior Técnico, Universidade de Lisboa, Lisboa 1000-029, Portugal, e-mail: sergio.jimenez@tecnico.ulisboa.pt, leonel.sousa@ist.utl.pt.

[2]Universidad de Extremadura, Departamento de Tecnología de los Computadores y de las Comunicaciones, Escuela Politécnica. Campus Universitario s/n, Cáceres 10003, España, e-mail: mavega@unex.es, jvicenteik@alumnos.unex.es.

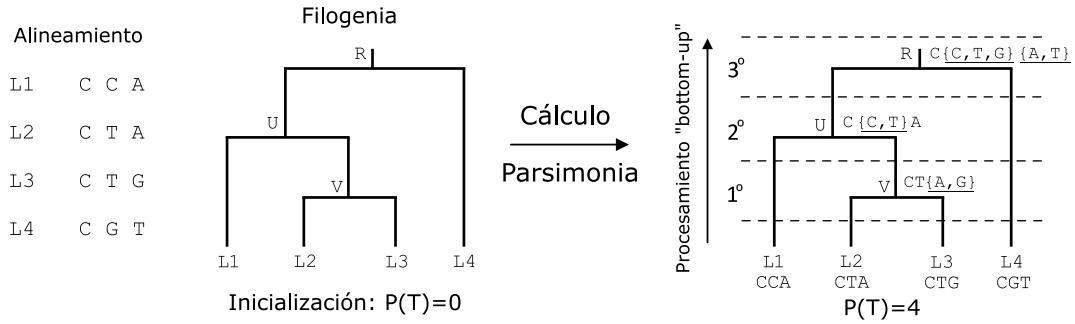


Fig. 1: Ejemplo de cálculo de $P(T)$. Tras inicializar los estados A_1, A_2, A_3 de las hojas con los valores observados en las secuencias de entrada, se procede con el procesamiento ascendente de los nodos internos. Para V, $A_1(V) = \mathrm{C}$ (dado que $A_1(L2) = A_1(L3) = \mathrm{C}$), $A_2(V) = \mathrm{T}$ ($A_2(L2) = A_2(L3) = \mathrm{T}$) y $A_3(V)$={A,G} (al ser $A_3(L2) = \mathrm{A}$!= $A_3(L3) = \mathrm{G}$, tomándose por tanto la unión). Estos pasos se repiten para U (considerando los conjuntos A_1, A_2, A_3 de sus hijos $L1$ y V) y, finalmente, para R (considerando los conjuntos de U y $L4$). El valor $P(T)$ resultante es 4, al haberse detectado intersecciones vacías en $A_3(V)$, $A_2(U)$, $A_2(R)$ y $A_3(R)$

II. Formulación del Problema

Las funciones objetivo filogenéticas proporcionan medidas de calidad biológica que guían el proceso de reconstrucción de hipótesis evolutivas [2]. Dichas hipótesis se representan mediante estructuras $T = (V, E)$ (árboles filogenéticos) que modelan las relaciones ancestro-descendiente entre los organismos descritos en el conjunto de nodos V mediante las ramas de E. Las hojas de V representan las especies cuya historia filogenética se pretende inferir, mientras que los nodos internos describen ancestros hipotéticos. A fin de reconstruir la topología filogenética, es necesario procesar un alineamiento múltiple de secuencias de tamaño $N \times M$, donde N es el número de especies a estudiar y M la longitud de secuencia. Los estados que puede tomar cada carácter de las secuencias vienen dados por un alfabeto Λ, el cual comprende en ADN los posibles nucleótidos {A,C,G,T}, *gaps* (-), valores desconocidos (?) y combinaciones.

La función de parsimonia tiene por objeto identificar los cambios de estado de carácter en las secuencias asociadas a especies emparentadas. La salida de dicha función es un valor entero que mide la cantidad de mutaciones observadas a lo largo de la topología, dándose preferencia a los árboles que minimicen dicho valor. Dada una filogenia $T = (V, E)$, su valor de parsimonia $P(T)$ puede expresarse como:

$$P(T) = \sum_{i=1}^{M} \sum_{(u,v)\in E} C(u_i, v_i), \qquad (1)$$

donde $(u, v) \in E$ representa la rama que enlaza a los nodos $u, v \in V$, $u_i, v_i \in \Lambda$ los valores de estado de u y v en el i-ésimo carácter de sus secuencias y $C(u_i, v_i)$ una función que cuantifica las mutaciones observadas ($C(u_i, v_i) = 1$ si $u_i \neq v_i$, o bien $C(u_i, v_i) = 0$ en caso de ser iguales). En este contexto, el cálculo de $P(T)$ puede llevarse a cabo mediante el algoritmo de Fitch [2]. La idea básica consiste en definir, para cada carácter $i = 1$ hasta M, un conjunto de posibles estados ancestrales A_i por nodo. Para cada nodo hoja l, $A_i(l)$ vendrá dado por el valor de estado l_i observado en el carácter i-ésimo de la secuencia correspondiente a l en el alineamiento. En el caso de los nodos internos, el algoritmo procesa la topología mediante una metodología *bottom-up*, calculando $A_i(u)$ para cada nodo u con hijos v, w como:

$$A_i(u) = \begin{cases} A_i(v) \cap A_i(w) & \text{si } A_i(v) \cap A_i(w) \neq \emptyset, \\ A_i(v) \cup A_i(w) & \text{si } A_i(v) \cap A_i(w) = \emptyset. \end{cases} \qquad (2)$$

La identificación de cambios de estado que precisa el cálculo de $P(T)$ puede realizarse mediante la Ecuación 2. En caso de que $A_i(v) \cap A_i(w)$ no sea $\emptyset$, esto implica que los hijos han heredado el estado de su ancestro, por lo que no se ha producido mutación durante el paso evolutivo. Por su parte, si $A_i(v) \cap A_i(w)$ es $\emptyset$, alguno de los hijos presenta una mutación en el carácter procesado. En este caso, el valor de $P(T)$ debe ser incrementado. La Figura 1 ejemplifica el cálculo de $P(T)$ bajo esta directriz.

Los costes computacionales de las funciones filogenéticas están en general gobernadas por la longitud M de las secuencias. La consideración de alineamientos con M grandes puede implicar tiempos de procesamiento significativos, los cuales se ven a su vez influidos por el número de especies N a considerar. De hecho, el valor de N no solo afecta al tamaño de la topología, sino que también al tamaño del espacio de búsqueda (con crecimientos exponenciales según el doble factorial $(2N - 5)!!$). La necesidad de dar confianza estadística a los análisis también afecta a los requisitos temporales del problema, al requerirse un número elevado de ejecuciones independientes con soporte *bootstrap*. Todos estos factores justifican el interés en explotar las oportunidades de paralelismo presentes en las funciones objetivo filogenéticas.

III. Aproximaciones GPU para Parsimonia

Con objeto de acelerar la función considerada, estudiaremos implementaciones del kernel de parsimonia utilizando distintos frameworks de programación GPU, concretamente CUDA, OpenCL y OpenACC.

A. Estructuras de Datos

En los diseños propuestos, el alineamiento de entrada viene representado por un array de caracteres

TABLA I: Codificación hexadecimal de estados

Estado	Valor	Estado	Valor
A (Adenina)	0x08	Y (C o T)	0x05
C (Citosina)	0x04	K (G o T)	0x03
G (Guanina)	0x02	V (A o C o G)	0x0E
T (Timina)	0x01	H (A o C o T)	0x0D
M (A o C)	0x0C	D (A o G o T)	0x0B
R (A o G)	0x0A	B (C o G o T)	0x07
W (A o T)	0x09	? (Desconocido)	0x0F
S (C o G)	0x06	- (*Gap*)	0x10

de longitud $N*M$, el cual almacena las secuencias en orden *row-major*. Los caracteres en este array están codificados mediante los valores hexadecimales mostrados en la Tabla I, considerando los posibles estados del alfabeto de ADN. Por su parte, los árboles filogenéticos a evaluar han sido implementados mediante la clase *TreeTemplate* de la librería BIO++ [8]. A partir de esta representación, se ha definido una estructura de datos separada para codificar de forma eficiente la topología en la GPU. Cada nodo interno viene dado por un tipo estructurado, denominado *fitch_node*, que contiene un short para representar el número de hijos del nodo, así como un array de enteros que almacena los identificadores de dichos hijos. El bit más significativo en estos identificadores indica si el hijo es un nodo interno (1) o un nodo hoja (0). Así, la topología en la GPU se implementa como un array de elementos *fitch_node*, organizado conforme al recorrido postorden del árbol para respetar las dependencias de datos del algoritmo.

B. Diseños CUDA y OpenCL

Los modelos de programación de CUDA [5] y OpenCL [6] se basan en la existencia de una CPU host que remite el procesamiento de tareas paralelas a la GPU. Mientras que la GPU se encarga de la ejecución del kernel, el host es responsable de realizar operaciones de soporte como es el caso de la reserva de buffers de memoria GPU y las transferencias de datos. La diferencia principal radica en el hecho de que CUDA está específicamente diseñado para operar con GPUs NVIDIA. Por su parte, OpenCL incluye una capa adicional de abstracción que le permite ejecutar kernels en otras arquitecturas paralelas.

El Algoritmo 1 presenta el diseño CUDA/OpenCL del kernel de parsimonia. La idea consiste en explotar el paralelismo de datos del algoritmo de Fitch, habida cuenta que el cómputo de los conjuntos de estados A_{tid} para un carácter *tid* es independiente de los cómputos relativos a cualquier otro carácter. Esto permite definir un kernel en el que cada hilo (CUDA) o *work-item* (OpenCL) con identificador *tid* opera sobre el *tid*-ésimo carácter de las secuencias, calculando así un valor de parsimonia parcial $P_{tid}(T)$.

Al comienzo de la ejecución, se obtienen los identificadores globales de los hilos (línea 1 en el Algoritmo 1). Además, cada hilo inicializa su valor de parsimonia parcial P_{tid} así como el array de estados A_{tid} donde se almacenará, en cada posición i, los valores calculados para el nodo i-ésimo de la topología

Algoritmo 1 Diseño CUDA / OpenCL del kernel

Entrada: `__global__` char* *secuencias* (secuencias de ADN),
`__constant__` fitch_node* *árbol* (filogenia a procesar),
`__constant__` int *long_sec* (longitud de secuencia),
`__constant__` int *num_internos* (número de nodos internos).
Salida: `__global__` int* *parsimonia_acc* (valores parciales de parsimonia acumulados).

```
1:  tid ← Obtener ID Global de Hilo /* blockIdx.x * blockDim.x
    + threadIdx.x en CUDA, get_global_id(0) en OCL */
2:  P_tid ← 0, A_tid ← 0
3:  /* Procesando la topología */
4:  para i = 1 hasta num_internos hacer
5:     valor_estado ← 0x1F /* Inicializando estado del nodo */
6:     para j = 1 hasta árbol[i].num_hijos hacer
7:        /* Leyendo datos del hijo j-ésimo */
8:        hijo_tipo ← árbol[i].hijo_identificador[j] & 0x80000000
9:        hijo_id ← árbol[i].hijo_identificador[j] & 0x7FFFFFFF
10:       si hijo_tipo = 0x80000000 entonces
11:          hijo_estado ← A_tid[hijo_id] /* Nodo interno */
12:       si no
13:          hijo_estado ← secuencias[long_sec*hijo_id+tid] /*
             Nodo hoja */
14:       fin si
15:       /* Aplicando operaciones de Fitch */
16:       fitch_result ← valor_estado & hijo_estado
17:       P_tid,valor_estado ← (fitch_result=0) ? {P_tid+1, va-
          lor_estado | hijo_estado} : {P_tid, fitch_result}
18:    fin para
19:    A_tid[i] ← valor_estado
20: fin para
21: /* Reducción paralela a nivel bloque / work-group */
22: __shared__ int* bloque_parsimonia ← 0
23: lid ← Obtener ID Local de Hilo /* threadIdx.x en CUDA,
    get_local_id(0) en OCL */
24: bid ← Obtener ID de Bloque /* blockIdx.x en CUDA,
    get_group_id(0) en OCL */
25: bloque_parsimonia[lid] ← P_tid
26: parsimonia_acc[bid] ← Reducción (bloque_parsimonia)
```

codificada (línea 2). El kernel lleva a cabo el procesamiento de los nodos de la topología a través de un bucle for (líneas 3–20). El cálculo de $A_{tid}(i)$ para un nodo i requiere la lectura de información sobre sus nodos hijos. Concretamente, es preciso obtener el tipo e identificador de cada hijo j (líneas 8–9), leyendo posteriormente el valor de estado del hijo en su *tid*-ésimo carácter 1) desde $A_{tid}(j)$ en el caso de que j fuese un nodo interno o 2) desde las secuencias de entrada en caso de que j sea un nodo hoja (líneas 10–14). A continuación se llevan a cabo los cálculos asociados a las operaciones de Fitch (líneas 15–17), en particular, operaciones de unión e intersección implementadas mediante instrucciones AND y OR. De acuerdo al resultado de estas operaciones, se actualiza el valor de parsimonia parcial del árbol, así como el valor de estado del nodo en procesamiento. El valor final de $A_{tid}(i)$ (línea 19) se consigue mediante la repetición de los pasos anteriores sobre cada hijo.

Seguidamente, el kernel procede a ejecutar una nueva iteración del bucle, procesando el siguiente nodo $i+1$ de la topología codificada. Una vez todos los nodos hayan sido procesados, cada hilo contendrá en su variable local P_{tid} el valor de parsimonia parcial para el *tid*-ésimo carácter. El kernel finaliza con la acumulación de estos valores a nivel de bloque / *work-group* (líneas 21–26), ejecutando una reducción paralela mediante el algoritmo «Reduce6» de [5].

Las tareas del host involucran: 1) la inicialización de la topología a partir de la estructura *TreeTemplate*, 2) la transferencia de dicha topología a la GPU mediante `cudaMemcpy` (CUDA) / `clEnqueueWriteBuffer` (OpenCL), 3) la invocación

del kernel, 4) la lectura de los resultados acumulados de parsimonia desde la GPU mediante `cudaMemcpy` / `clEnqueueReadBuffer` y 5) la reducción final de los valores acumulados para obtener el resultado final de $P(T)$. Esta implementación hace uso de operaciones asíncronas y *streams* CUDA / colas de comandos OpenCL para solapar la ejecución del kernel con operaciones independientes en el lado host. Esto permite a su vez la ejecución concurrente de llamadas al kernel para distintos árboles filogenéticos.

C. Diseño OpenACC

OpenACC se define como un estándar de alto nivel, basado en directivas, para la programación de aplicaciones en aceleradores hardware [7]. El paralelismo presente en la aplicación es expuesto mediante el uso de directivas y funciones sobre la versión secuencial, facilitando así la especificación de tareas paralelas a ejecutar en un coprocesador.

El Algoritmo 2 presenta el diseño OpenACC de la función de parsimonia. Se emplea la directiva `#pragma acc kernels` para indicar al compilador la región de código a ejecutar en la GPU (línea 1 en al Algoritmo 2). Mediante las cláusulas `copyin` y `copyout`, podemos especificar las variables a comunicar a la GPU con anterioridad a la ejecución del kernel (topología y número de nodos internos) así como las variables a devolver tras terminar los cálculos paralelos (valor de parsimonia). Dado que las secuencias representan la estructura de datos de mayor tamaño, se procede a su transferencia al comienzo de la aplicación mediante `#pragma acc enter data`, de tal manera que podemos emplear la cláusula `present` en `#pragma acc kernels` para indicar que esta estructura está ya almacenada en la memoria de la GPU. Esto permite evitar transferencias redundantes para invocaciones independientes del kernel.

El bucle principal del algoritmo de Fitch itera sobre la longitud de secuencia para calcular las parsimonias parciales de cada carácter i (líneas 4–23). Se introduce la directiva `#pragma acc loop` para paralelizar los cómputos del bucle (línea 3). La cláusula `private` permite especificar copias privadas del array de estados A_i para cada hilo, introduciéndose a su vez la cláusula `reduction` para acumular los valores parciales de parsimonia P_i. Cada nodo j en la topología es procesado dentro del bucle paralelo, leyendo los datos de sus k hijos (líneas 9–16) para calcular $A_i(j)$ y actualizar la parsimonia parcial P_i (líneas 17–19, 21). El valor final de $P(T)$ obtenido tras la reducción constituye la salida del algoritmo (línea 24). La propuesta incluye los mecanismos de asincronía definidos en OpenACC. Concretamente, empleamos la cláusula `async` en `#pragma acc kernels` para conseguir solapamiento entre ejecuciones de kernel e inicializaciones / transferencias de datos.

Algoritmo 2 Diseño OpenACC del kernel

```
Entrada: char* secuencias (secuencias de ADN),
    const __restrict__ fitch_node* árbol (filogenia a procesar),
    const int long_sec (longitud de secuencia),
    const int num_internos (número de nodos internos).
Salida: int parsimonia_acc (valor de parsimonia).
1:  #pragma acc kernels copyin (árbol, num_internos) present
    (secuencias, long_sec) copyout (parsimonia_acc)
2:  P_i ← 0
3:  #pragma acc loop independent private(A_i) reduction(+:P_i)
4:  para i = 1 hasta long_sec hacer
5:    /* Procesando la topología */
6:    para j = 1 hasta num_internos hacer
7:      valor_estado ← 0x1F /* Inicializando estado */
8:      para k = 1 hasta árbol[j].num_hijos hacer
9:        /* Leyendo datos del hijo k-ésimo */
10:       hijo_tipo ← árbol[j].hijo_identificador[k] &
          0x80000000
11:       hijo_id ← árbol[j].hijo_identificador[k] &
          0x7FFFFFFF
12:       si hijo_tipo = 0x80000000 entonces
13:         hijo_estado ← A_i[hijo_id] /* Nodo interno */
14:       si no
15:         hijo_estado ← secuencias[long_sec*hijo_id+i] /*
            Nodo hoja */
16:       fin si
17:       /* Aplicando operaciones de Fitch */
18:       fitch_result ← valor_estado & hijo_estado
19:       P_i,valor_estado ← (fitch_result=0) ? {P_i+1, va-
          lor_estado | hijo_estado} : {P_i, fitch_result}
20:     fin para
21:     A_i[j] ← valor_estado
22:   fin para
23: fin para
24: parsimonia_acc ← P_i /* P_i contiene la reducción */
```

IV. Resultados Experimentales

Esta sección da cuenta de la evaluación experimental de las aproximaciones GPU consideradas para acelerar la función filogenética de parsimonia. Para ello, hemos empleado tres GPUs de NVIDIA, concretamente: 1) Tesla K40m, compuesta por 15 *streaming multiprocessors* (SMs) y un total de 2880 cores CUDA bajo arquitectura Kepler a 0,75GHz, 2) Tesla K20Xm, compuesta por 14 SMs (2688 cores CUDA) con arquitectura Kepler a 0,73GHz, y 3) GeForce GTX TITAN X, con 24 SMs (3072 cores CUDA) con arquitectura Maxwell a 1,08GHz. Estas GPUs están incluidas en una infraestructura hardware con 2 procesadores Intel Xeon E5-2630v3 (16 cores físicos, 32 cores lógicos mediante *hyperthreading*) a 2,40GHz y 80GB de RAM. El software instalado incluye Ubuntu 14.04LTS, CUDA 7.5, GCC 4.9.3 con NVIDIA OpenCL 1.2 y el compilador PGI 17.4 (OpenACC). A su vez, se ha empleado en la comparativa un coprocesador Intel Xeon Phi 7120P (61 cores, 60 útiles para cómputo) a 1,33GHz con 16GB de RAM.

Se han utilizado seis conjuntos de datos reales de ADN a fin de examinar el rendimiento de los diseños propuestos en distintos tamaños del problema:

1. M156x119750: datos de salmonela, 156 secuencias de longitud 119750 caracteres [9].
2. M50x42456: datos de enterobacterias, 50 secuencias de longitud 42456 caracteres [10].
3. M169x24251: datos de mamíferos, 169 secuencias de longitud 24251 caracteres [11].
4. M1459x8610: datos de VIH1, 1459 secuencias de longitud 8610 caracteres [12].
5. M55x4675: datos de anfibios cecílidos, 55 secuencias de longitud 4675 caracteres [13].
6. M500x759: datos del gen rbcL, 500 secuencias de longitud 759 caracteres [14].

Con objeto de garantizar la fiabilidad estadística del estudio, se han realizado 31 ejecuciones indepen-

dientes de la aplicación por experimento. Cada ejecución involucró el cálculo de $P(T)$ para 2000 árboles filogenéticos (generados a partir de muestras *bootstrap* del alineamiento de entrada [2]).

A. Caracterización del Kernel

En primer lugar, caracterizaremos el comportamiento del kernel a fin de comprender cómo la función de parsimonia explota los recursos GPU. Para ello se han empleado las herramientas de análisis *nvprof* y *visual profiler* de NVIDIA, usando la versión CUDA de la aplicación en Tesla K40m.

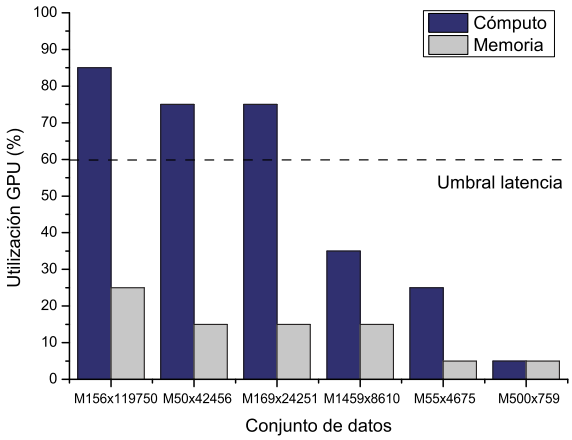
La Figura 2(a) muestra cómo los niveles de utilización de unidades funcionales y memorias evolucionan en los conjuntos de datos estudiados. En base a estos resultados, es posible establecer una correspondencia entre los valores de longitud de secuencia y el comportamiento del kernel. La mayor explotación de recursos GPU es observada en instancias con más de 100 mil nucleótidos por secuencia, representando un escenario de cómputo intensivo. Moviéndonos al rango 10-100 mil, se verifica que el kernel sigue presentando un comportamiento de tipo *compute-bound*, implicando pues un uso significativo de las unidades aritméticas (enteros) de la GPU. Sin embargo, las oportunidades reducidas de paralelismo asociadas a las instancias con longitud corta de secuencia dan lugar a escenarios de ejecución gobernados por restricciones de latencia, dándose el caso peor con longitudes menores de mil nucleótidos por secuencia.

Para complementar esta caracterización, la Figura 2(b) presenta los porcentajes temporales contribuidos por los cálculos del kernel y las transferencias de datos en la aplicación. En término medio, los cómputos del kernel representan un 97 % del tiempo global de ejecución, mientras que las comunicaciones entre el host y la GPU dan lugar a porcentajes temporales por debajo del 3 %. Por lo tanto, las transferencias de datos no actúan como un factor limitante en el rendimiento de esta aplicación. Como ilustración, las transferencias en el caso de las instancias con mayor número de secuencias (M1459x8610) y longitud de secuencia (M156x119750) apenas representan el 0,8 / 1,1 % del tiempo de ejecución, respectivamente.

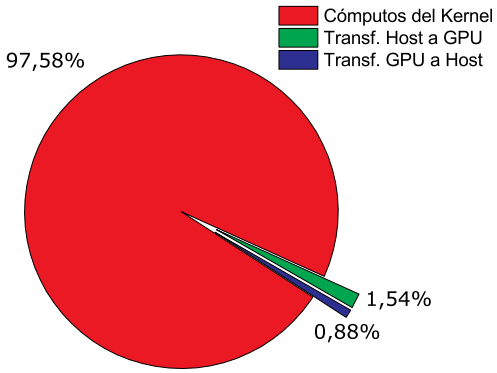
B. Comparativa de Diseños GPU

La Tabla II proporciona la comparativa de tiempos de ejecución de los versiones CUDA, OpenCL y OpenACC de la aplicación. En concreto, damos cuenta de los tiempos de transferencia y de cómputos del kernel para las ejecuciones de tiempo de ejecución mediano. A su vez, la Tabla III resume las instrucciones PTX generadas por cada aproximación.

Podemos observar que la versión CUDA obtiene los mejores resultados en todos los escenarios evaluados. Este diseño da lugar a mejoras generales tanto en los tiempos de kernel como de transferencia, minimizando así el tiempo de ejecución en las distintas instancias consideradas. En términos prácticos, CUDA consigue mejoras temporales medias de 13,7 % / 65,7 % (M156x119750), 24,3 % / 60,3 %



(a) Utilización de unidades funcionales y memoria



(b) Porcentajes temporales por operación

Fig. 2: Caracterización del kernel de parsimonia

(M50x42456), 17,6 % / 48,5 % (M169x24251), 2,2 % / 42,8 % (M1459x8610), 43,3 % / 54,1 % (M21119_55) y 9,7 % / 35,2 % (M500x759) sobre OpenCL y OpenACC, respectivamente. Este comportamiento se ve a su vez reflejado en las métricas de eficiencia SM y ejecución a nivel *warp*, verificándose en K40m valores medios de 95,6 % (SM) y 91,2 % (*warp*).

Por su parte, el diseño OpenCL proporciona los segundos mejores tiempos de ejecución en todos los escenarios analizados. La comparativa con OpenACC da cuenta de aceleraciones de hasta 2,4x (K40m), 2,3x (K20Xm) y 3x (TITAN X) al usar OpenCL. Aunque OpenCL no alcanza los resultados paralelos de CUDA, las diferencias en tiempo de ejecución resultan menos notables (menos del 15 %) en la instancia con mayor longitud de secuencia (M156x119750) y en aquellas con mayor número de secuencias (M500x759 y M1459x8610). Puede observarse en la Tabla III que tanto CUDA como OpenCL generan distribuciones de instrucciones PTX similares, dándose el caso de que OpenCL precisa de un número ligeramente mayor de movimientos de datos en registros (MOV), cargas (LD) y operaciones aritméticas (ADD). Sin embargo, la versión OpenCL presenta mayores requisitos de registros (40 registros de 32 bits, en comparación a los 32 usados por CUDA). A pesar de estas diferencias, los resultados en M156x119750 y M1459x8610 sugieren que OpenCL representa una opción válida para acometer la ace-

TABLA II: Tiempos de transferencia (T_{com_acc}), kernel (T_{ker_acc}) y ejecución (T_{exec_ov}, con solapamiento) en milisegundos, ejecuciones de tiempo mediano (2000 árboles filogenéticos procesados por ejecución)

	CUDA			OpenCL			OpenACC		
				Tesla K40m					
Instancia	T_{com_acc}	T_{ker_acc}	T_{exec_ov}	T_{com_acc}	T_{ker_acc}	T_{exec_ov}	T_{com_acc}	T_{ker_acc}	T_{exec_ov}
M156x119750	15,915	2676,036	**1494,484**	16,271	2900,918	1587,586	22,638	4482,421	3804,667
M50x42456	8,028	299,264	**212,713**	8,213	354,291	262,698	17,396	588,313	547,467
M169x24251	9,766	781,176	**499,994**	10,801	887,933	577,187	18,676	1470,240	952,455
M1459x8610	21,716	4056,261	**2630,666**	34,267	4225,927	2742,374	37,030	7739,716	4522,540
M55x4675	6,928	116,067	**125,003**	7,251	262,003	216,057	16,833	295,637	272,858
M500x759	10,031	988,266	**739,065**	14,978	1151,671	814,225	22,545	1680,830	1215,605
				Tesla K20Xm					
M156x119750	17,282	2816,869	**1524,103**	20,266	3627,217	1950,854	23,608	5016,601	4383,438
M50x42456	7,826	304,313	**215,133**	7,951	363,509	273,596	20,636	628,246	600,896
M169x24251	10,254	799,142	**506,049**	10,455	914,512	686,225	20,923	1580,958	1032,035
M1459x8610	23,254	4258,498	**2695,264**	32,672	4331,688	2746,011	39,447	7900,515	4883,622
M55x4675	6,813	128,435	**136,489**	7,090	291,512	238,756	18,061	315,196	290,421
M500x759	10,330	1131,162	**809,609**	15,003	1334,765	921,530	27,015	1915,001	1326,436
				GTX TITAN X					
M156x119750	15,087	833,088	**527,836**	13,862	960,383	608,381	18,119	2209,639	1831,816
M50x42456	6,482	156,136	**129,922**	7,204	229,498	192,802	15,970	303,634	291,493
M169x24251	9,706	479,780	**330,792**	8,821	514,245	381,500	18,409	792,030	625,791
M1459x8610	16,200	2965,442	**1982,250**	25,298	3007,448	1993,854	25,175	5409,487	3404,137
M55x4675	6,336	100,993	**105,689**	6,541	226,110	191,625	16,883	287,972	235,133
M500x759	7,939	769,409	**645,375**	10,878	868,079	699,881	20,119	1038,693	888,297

TABLA III: Comparativa de PTX CUDA, OpenCL (OCL) y OpenACC (OACC)

Inst.	CUDA	OCL	OACC	Inst.	CUDA	OCL	OACC
MOV	21	24	17	ADD	20	22	31
LD	13	15	24	SUB	1	1	4
ST	5	5	10	MUL	6	6	1
CVT	3	3	7	AND	5	5	3
CVTA	2	2	5	OR	1	1	1
BRA	14	14	19	SHL/R	3	3	3
SETP	13	13	17	MAD	2	2	1
SELP	2	2	2	BAR	3	3	2

leración GPU de funciones objetivo en escenarios de evaluación complejos (con la ventaja de permitir mayor heterogeneidad de recursos que CUDA).

Finalmente, OpenACC representa el último escalón en la comparativa. El análisis de su código PTX da cuenta de diferencias apreciables con respecto a CUDA y OpenCL, observándose aumentos significativos en el número de 1) instrucciones aritméticas (ADD y SUB) y 2) operaciones de carga y almacenamiento (LD y ST). De hecho, la explotación de unidades funcionales aritméticas dista de los resultados de CUDA, al conseguir OpenACC niveles de utilización bajos y medios según la herramienta *nvprof*. Es más, el número de operaciones LD y ST sugiere dificultades en el modo en que el kernel OpenACC gestiona los accesos a las memorias de la GPU, representando las dependencias de memoria el factor con mayor impacto en la ejecución. Todo ello da lugar a un empeoramiento notable del rendimiento (por encima del 10 %–20 % que suele esperarse al emplear directivas OpenACC [7]). No obstante, la minimización en los tiempos de desarrollo (dos semanas frente a los dos meses requeridos por los diseños CUDA y OpenCL) representa una ventaja a considerar en contextos con tiempos de programación restringidos.

C. Comparativa de Arquitecturas GPU

A continuación, analizaremos el rendimiento observado con CUDA en los distintos dispositivos GPU considerados mediante el uso de dos métricas. En primer lugar, hemos medido el rendimiento de cómputo teniendo en cuenta el número de operaciones de Fitch realizadas durante la ejecución del kernel. Esta métrica, que hemos denominado *Giga-Fitch Operations per Second* (GFOS), se define como:

$$GFOS = ((N_{internos} * M + P(T))/t)/10^9, \quad (3)$$

donde $N_{internos}$ representa el número de nodos internos de la topología, M la longitud de secuencia, $P(T)$ el valor de parsimonia y t el tiempo de ejecución para una invocación del kernel. En segundo lugar, consideraremos el uso efectivo de ancho de banda de memoria en Gigabytes por segundo (BW) [15]:

$$BW = ((R_B + W_B)/t)/10^9, \quad (4)$$

donde R_B representa el número de bytes leídos por el kernel y W_B el número de bytes escritos.

La Tabla IV proporciona los valores de estas métricas para K40m, K20Xm y TITAN X en las ejecuciones de tiempo mediano. Para el caso de los GFOS, las columnas «Inferior» se refieren a los GFOS observados en la invocación del kernel que operó sobre la topología con menor valor de $P(T)$, mientras que las columnas «Superior» son sus contrapartidas sobre la topología con mayor valor de $P(T)$.

La evaluación de las GPUs Kepler muestra que K40m permite obtener una mejora media de 4,3 % en GFOS y 11,8 % en BW sobre K20Xm. Por otra parte, el mejor dispositivo en esta comparativa viene dado por la TITAN X Maxwell, cuyo uso conlleva un incremento significativo del 60,6 % en GFOS y 60,8 % en BW sobre K40m. Las diferencias en GFOS entre las GPUs Kepler y TITAN X crecen notablemente al aumentar la longitud de secuencia, resultando en mejoras de entre 14,5 %–32,7 % en las instancias gobernadas por latencias y 51,2 %–183 % en las intensivas en cómputo. Al verse dominado el kernel de

TABLA IV: Rendimiento de cómputo (*Giga-Fitch Operations per Second*, GFOS) y ancho de banda (BW)

	Tesla K40m			Tesla K20Xm			GTX TITAN X		
	GFOS		BW	GFOS		BW	GFOS		BW
Instancia	Inferior	Superior	Valor	Inferior	Superior	Valor	Inferior	Superior	Valor
M156x119750	49,935	49,956	23,788 GB/s	48,965	48,985	18,891 GB/s	**141,384**	**141,442**	**45,199 GB/s**
M50x42456	41,816	41,844	18,477 GB/s	41,346	41,373	16,837 GB/s	**68,464**	**68,509**	**29,891 GB/s**
M169x24251	33,497	33,497	11,750 GB/s	33,096	33,097	11,115 GB/s	**50,631**	**50,631**	**15,915 GB/s**
M1459x8610	19,623	19,623	6,373 GB/s	19,152	19,153	5,921 GB/s	**26,042**	**26,042**	**8,422 GB/s**
M55x4675	8,546	8,546	3,899 GB/s	7,827	7,827	3,568 GB/s	**10,108**	**10,108**	**5,353 GB/s**
M500x759	2,090	2,091	0,710 GB/s	1,908	1,908	0,631 GB/s	**2,394**	**2,394**	**0,968 GB/s**

TABLA V: Tiempos CPU / Xeon Phi (en segundos) y aceleraciones GPU (FA, con TITAN X)

Tiempos de ejecución				
Instancia	Serie	1xXeon	2xXeon	XeonPhi
M156x119750	170,878	5,051	2,481	3,367
M50x42456	27,758	0,946	0,443	0,523
M169x24251	70,570	2,133	0,912	0,990
M1459x8610	233,202	8,395	4,296	3,384
M55x4675	3,826	0,275	0,274	0,395
M500x759	5,658	0,803	0,649	0,786
Instancia	FA(GPU, Serie)	FA(GPU, 1xXeon)	FA(GPU, 2xXeon)	FA(GPU, XeonPhi)
M156x119750	323,733x	9,569x	4,700x	6,379x
M50x42456	213,653x	7,282x	3,410x	4,029x
M169x24251	213,336x	6,447x	2,758x	2,992x
M1459x8610	117,645x	4,235x	2,167x	1,707x
M55x4675	36,201x	2,599x	2,590x	3,742x
M500x759	8,767x	1,244x	1,005x	1,218x

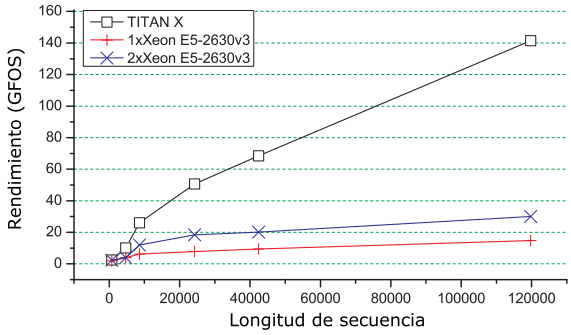
parsimonia por las operaciones realizadas sobre tipos enteros, la aplicación se ve beneficiada por la reducción de latencias aritméticas básicas de la arquitectura Maxwell. Consecuentemente, es posible verificar un rendimiento más escalable a medida que aumentamos la longitud de secuencia considerada.

D. Comparativa con otras Arquitecturas Paralelas

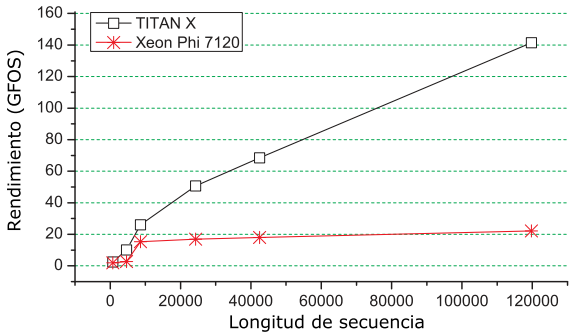
Finalmente, abordaremos comparaciones con otras plataformas y alternativas paralelas. La parte superior de la Tabla V introduce los tiempos de ejecución medianos observados en la CPU Intel Xeon-2630v3 y el Xeon Phi 7120P con OpenCL. La columna *Serie* hace referencia a los tiempos de la implementación serie CPU, *1xXeon* la versión CPU paralela con 16 unidades de cómputo OpenCL, *2xXeon* la versión CPU paralela con 32 unidades y *XeonPhi* la versión paralela Xeon Phi con 240 unidades de cómputo. Las aceleraciones (FA) conseguidas sobre estos dispositivos al usar el diseño GPU (CUDA, con TITAN X) son presentadas en la parte inferior de la Tabla V.

Según los resultados obtenidos, la aproximación GPU es capaz de mejorar los tiempos de las configuraciones CPU y Xeon Phi en todos los conjuntos de datos analizados. Con respecto a la configuración multicore *1xXeon*, se observan aceleraciones de hasta 4,2x en las instancias gobernadas por latencias, mientras que en las intensivas en cómputo es posible conseguir aceleraciones de hasta 9,6x. El diseño GPU mejora también los tiempos de ejecución con respecto a la configuración multiprocesador *2xXeon*, así como al co-procesador Intel Xeon Phi.

La Figura 3 muestra la evolución de rendimiento de cómputo (en GFOS) en TITAN X y las configuraciones CPU / Xeon Phi para longitudes crecientes



(a) GPU vs. CPU



(b) GPU vs. Xeon Phi

Fig. 3: Comparativa de rendimiento (en GFOS, *Giga-Fitch Operations per Second*)

de secuencia. A partir de estos valores, se han calculado las líneas de tendencia que mejor aproximan la escalabilidad de cada plataforma. En el caso de la GPU, su escalabilidad viene aproximada por la función potencial $3,31x^{0,82}$, donde x es la longitud de secuencia, con un coeficiente R2 del 0,99. Por su parte, las configuraciones CPU pueden aproximarse con las funciones $2,22x^{0,40}$ (*1xXeon*, R2=0,99) y $2,66x^{0,54}$ (*2xXeon*, R2=0,92). Finalmente, la Xeon Phi representa un caso intermedio con respecto a las configuraciones CPU: $4,28ln(x)+1,88$ (R2 = 0,84).

Estas funciones sugieren que las diferencias en rendimiento entre el diseño GPU y los otros dispositivos se irán incrementando en escenarios con mayor longitud de secuencia. Este hecho presenta una importancia significativa desde un plano biológico, dada la mejora en calidad filogenética que supone la consideración de secuencias grandes en el análisis y el consecuente interés creciente en el procesamiento eficiente de las mismas. También resultan interesantes los resultados en longitud limitada, al dar cuenta de cómo los avances en latencias operacionales pueden permitir ir un paso más allá incluso en escenarios

TABLA VI: Tiempos de Parsimonator (en segundos) y aceleraciones GPU (FA, con TITAN X)

	Tiempos Parsimonator		Aceleraciones GPU	
Instancia	OMP+ SSE3	OMP+ AVX	FA(GPU, OMP+SSE3)	FA(GPU, OMP+AVX)
M156x119750	3,037	2,771	5,754x	5,250x
M50x42456	0,776	0,753	5,971x	5,796x
M169x24251	2,665	2,535	8,057x	7,663x
M1459x8610	20,448	17,145	10,316x	8,649x
M55x4675	0,849	0,793	8,033x	7,501x
M500x759	6,238	6,055	9,665x	9,382x

con oportunidades más reducidas de paralelismo.

Para concluir, la Tabla VI presenta una comparativa con el método Parsimonator [3], el cual incluye una implementación CPU paralela de la función de parsimonia basada en OpenMP (OMP) e instrucciones SIMD. Las columnas 2 y 3 de esta tabla introducen los tiempos de ejecución medianos de Parsimonator en nuestra configuración hardware, usando 32 hilos OpenMP e instrucciones SSE3/AVX. Los factores de aceleración conseguidos por nuestro diseño GPU (CUDA en TITAN X) pueden encontrarse en las columnas 4 y 5. Es posible observar aceleraciones efectivas en los intervalos 5,8x–10,3x y 5,3x–9,4x sobre las versiones SS3+OpenMP y AVX+OpenMP de Parsimonator. Estas mejoras temporales se aprecian más notablemente en los conjuntos de datos con un número elevado de secuencias (M500x759 y M1459x8610), lo cual sugiere la bondad de nuestro diseño a la hora de procesar topologías de tamaño creciente.

V. Conclusiones

Este trabajo ha abordado el análisis de tecnologías GPGPU para acelerar funciones objetivo. El estudio tenía por objeto examinar las oportunidades que estos dispositivos conllevan a la hora de paralelizar los procesos de evaluación en problemas de optimización, así como establecer una relación entre las caracteríticas de la función estudiada y la explotación de recursos GPU. Como caso de estudio, se han diseñado kernels CUDA, OpenCL y OpenACC para paralelizar una importante función objetivo bioinformática, la función filogénetica de parsimonia.

Mediante experimentación sobre seis conjuntos de datos reales de ADN, hemos efectuado la caracterización del kernel de parsimonia y discutido la influencia de la longitud de secuencia sobre la utilización de recursos GPU. Seguidamente, la evaluación de modelos de programación nos ha permitido distinguir las ventajas e inconvenientes principales de cada diseño, obteniéndose los mejores resultados paralelos mediante CUDA. A su vez, hemos dado cuenta de los beneficios que implica la mejora en latencias operacionales y capacidades de cómputo en arquitecturas GPUs modernas, dando lugar a mejoras sobre propuestas previas y otras plataformas paralelas tanto en instancias intensivas en cómputo como dominadas por latencias. De hecho, la evolución del rendimiento confirma el papel de importancia que las GPUs pueden asumir en la aceleración de funciones objetivo en escenarios de optimización presentes y futuros.

Como trabajo futuro, pretendemos seguir explorando el diseño de aproximaciones paralelas para mejorar procesos de optimización complejos. En primer lugar, realizaremos la adaptación de los kernels GPU de parsimonia a otros escenarios biológicos, como es el caso de las proteínas, abordando los problemas asociados al procesamiento de tipos de datos más complejos. Examinaremos también otras plataformas de cómputo, como procesadores ARM y FPGAs, y diseños heterogéneos que engloben una amplia variedad de recursos hardware. Finalmente, se estudiará la aplicación de estas estrategias a otros problemas.

Agradecimientos

Este trabajo ha sido parcialmente financiado por la AEI (Agencia Estatal de Investigación, España) y el FEDER (Fondo Europeo de Desarrollo Regional, Unión Europea), bajo el proyecto TIN2016-76259-P (proyecto PROTEIN), así como por la FCT (Fundação para a Ciência e a Tecnologia, Portugal) con referencia UID/CEC/50021/2019. Gracias también a la Junta de Extremadura y el FEDER por la ayuda GR18090 otorgada al grupo de investigación TIC015. Sergio Santander-Jiménez agradece a la FCT su contrato postdoctoral SFRH/BPD/119220/2016.

Referencias

[1] M. Nobile, P. Cazzaniga, A. Tangherloni, and D. Besozzi, "Graphics processing units in bioinformatics, computational biology and systems biology," *Brief. Bioinform.*, vol. 18, no. 5, pp. 870–885, 2017.

[2] P. Lemey, M. Salemi, and A. Vandamme, *The Phylogenetic Handbook: A Practical Approach to Phylogenetic Analysis and Hypothesis Testing*, Cambridge Univ. Press, Cambridge, UK, 2009.

[3] N. Alachiotis and A. Stamatakis, "FPGA Acceleration of the Phylogenetic Parsimony Kernel?," in *Proc. of FPL 2011*. 2011, pp. 417–422, IEEE.

[4] S. Santander-Jiménez, A. Ilic, L. Sousa, and M. A. Vega-Rodríguez, "Accelerating the Phylogenetic Parsimony Function on Heterogeneous Systems," *Concurr. Comp. Pract. E.*, vol. 29, no. 8, pp. 1–15, 2017.

[5] N. Wilt, *The CUDA Handbook: A Comprehensive Guide to GPU Programming*, Addison Wesley, Pearson, NJ, USA, 2013.

[6] D. R. Kaeli, P. Mistry, D. Schaa, and D. P. Zhang, *Heterogeneous Computing with OpenCL 2.0*, Morgan Kaufmann Publishers, MA, USA, 2015.

[7] R. Farber, *Parallel Programming with OpenACC - 1st Edition*, Morgan Kaufmann Publishers, MA, USA, 2017.

[8] L. Guéquen et al., "Bio++: efficient extensible libraries and tools for computational molecular evolution," *Mol. Biol. Evol.*, vol. 30, no. 8, pp. 1745–1750, 2013.

[9] R. E. Timme et al., "Phylogenetic diversity of the enteric pathogen Salmonella enterica subsp. enterica inferred from genome-wide reference-free SNP characters," *Genome Biol. Evol.*, vol. 5, no. 11, pp. 2109–2123, 2013.

[10] F. Husník, T. Chrudimský, and V. Hypša, "Multiple origins of endosymbiosis within the Enterobacteriaceae (γ-Proteobacteria): convergence of complex phylogenetic approaches," *BMC Biology*, vol. 9, no. 87, pp. 1–17, 2011.

[11] R. Meredith et al., "Impacts of the Cretaceous Terrestrial Revolution and KPg Extinction on Mammal Diversification," *Science*, vol. 334, no. 6055, pp. 521–524, 2011.

[12] "HIV Sequence Database," `http://www.hiv.lanl.gov/`, 2005.

[13] D. San Mauro et al., "Life-history evolution and mitogenomic phylogeny of caecilian amphibians," *Mol. Phylogenet. Evol.*, vol. 73, no. 1, pp. 177–189, 2014.

[14] M. W. Chase et al., "Phylogenetics of seed plants: An analysis of nucleotide sequences from the plastid gene rbcl," *Annals of the Missouri Botanical Garden*, vol. 80, no. 3, pp. 528–580, 1993.

[15] Nvidia, *CUDA C Best Practices Guide*, 2017.

Un nuevo enfoque para la visualización de datos de metilación del ADN: paralelización de la transformada wavelet en la GPU

Lisardo Fernández, Mariano Pérez y Juan M. Orduña[1]

Resumen— **En los últimos años se han propuesto diversos enfoques estadísticos para la identificación de regiones diferencialmente metiladas entre grupos de muestras de ADN, a partir de los datos ofrecidos por diferentes herramientas de anánalis de metilación. Sin embargo, estos enfoques no ofrecen una exploración interactiva de estas regiones, y añaden una carga computacional muy alta cuando se analizan conjuntos de datos muy grandes. En este trabajo proponemos un nuevo enfoque, consistente en la contrucción de una señal de metilación a partir de los resultados de metilación del ADN, y la aplicación de la transformada wavelet a dicha señal de metilación, para poder visualizar la señal de metilación a diferentes escalas. Además, proponemos la paralelización de la transformación wavelet así como la visualización de la señal de metilación desde los datos en la GPU. La evaluación prestaciones muestra que este enfoque permite la visualización interactiva de diferentes señales de metilación con diferentes niveles de resolución. En este sentido, también se puede utilizar para la visualización precisa de regiones diferencialmente metiladas de modo flexible, amigable y con un tiempo de ejecución interactivo.**

Palabras clave— **Análisis de metilación del ADN, Transformada Wavelet, Computación con GPU, Visualización con GPU.**

I. Introducción

La metilación del ADN consiste en la adhesión de un grupo metilo (CH_3) a una citosina, formando un enlace $5mC$. La hidroximetilación del ADN (un subconjunto de las posibles metilaciones del ADN) consiste en la oxidación del enlace $5mC$ en los dinucleótidos CpG para formar 5-hidroximetilcitosina ($5hmC$). La metilación del ADN es uno de los procesos epigenéticos que aparenta jugar un papel relevante en la regulación genética [1]. De hecho, aparenta ser relevante en el desarrollo de enfermedades complejas cono la hipertensión, la obesidad, el cáncer o la diabetes mellitus tipo 2 (DM2) [2]. Por ello, el análisis de la metilación se ha convertido en una importante vía en el estudio de la salud humana.

El análisis de metilación del ADN requiere tratamientos específicos que modifiquen su secuencia, así como herramientas software. Los datos de metilación se pueden obtener a partir de la secuenciación con bisulfito, técnica que proporciona mapas exhaustivos de metilación a nivel de resolución de par de base [3]. El tratamiento con bisulfito conviente las citosinas (Cs) no metiladas en timinas (Ts), cambio que mantienen después de la secuenciación del ADN, mientras que las citosinas metiladas (5mCs) se mantienen invariables. A partir del alineamiento y comparación de las *reads* secuenciadas con bisulfito con la secuencia del genoma de referencia, es posible inferir patrones de metilación del ADN con resolución de par de base. Las muestras hidroximetiladas se pueden obtener mediante dos métodos distintos: Tab-Seq (del inglés *Ten-eleven translocation (TET) Assisted Bisulfite Sequencing (TAB-Seq)* [4], [5]), que transforma en Timinas (T) las citosinas metiladas ($5mC$) y las no metiladas (C), mientras que mantiene como citosinas aquellas citosinas que están hidroximetiladas ($5hmc$); y el médtodo oxBs-seq (del inglés *oxidative bisulphate sequencing (oxBs-seq)* [6]. En cuanto a herramientas software, se han desarrollado muchas propuestas, como la ampliamente extendida Bismark [7] o la herramienta más reciente HPG-Methyl [8], [9]. Estas herramientas proporcionan información a nivel de base para cada secuencia muestreada (o *read*). Sin embargo, todas las herramientas software devuelven el resultado del análisis como ficheros de texto en formato *Secuence Alignment Map* (SAM) o en formato *Binary Alignment Map* (BAM), ficheros que suelen ocupar decenas o centenas de GBytes. Estos resultados en sí mismos no son útiles para los investigadores biomédicos, que necesitan comparar la información de metilación a diferentes escalas (segmentos de ADN, islas CpG (secuencia de citosina seguida de guanina), regiones codificantes (regiones de ADN que codifican funciones biológicas), cromosomas, etc), además de la resolución a nivel de base. Por ello, se han propuesto también múltiples herramientas para el procesamiento, visualización de los resultados de metilación e identificación de regiones diferencialmente metiladas (DMRs, del inglés *Differentielly Methyated Region*) [10], [11], [12], [13], [14], [15]. Sin embargo, estas herramientas se basan en técnicas estadísticas, añadiendo una elevada carga computacional sobre los enormes ficheros BAM o SAM. Como resultado, el tiempo requerido para la ejecución del proceso de análisis sobre estos ficheros es muy alto, añadiendo un retraso especialemente largo al proceso desde la extracción de las muestras del secuenciador de ADN hasta el momento de entregar información relevante al investigador biomédico.

En un trabajo previo, exploramos un nuevo enfoque basado en la generación de una señal de metilación a partir de los resultados de metilación del ADN para, sobre ella, aplicar la transformación

[1]Dpto. de Informática, Universidad de Valencia, e-mail: {lisardo.fernandez,mariano.perez,juan.orduna}@uv.es

wavelet Haar y visualizar el resultado de la metilación a la escala requerida [16]. Este enfoque proporciona resultados de visualización similares a otras herramientas existentes, pero, además, muestra la señal con diferentes niveles de resolución. De este modo, es posible identificar y visualizar los DMRs con una velocidad mucho mayor. En este trabajo proponemos una paralelización eficiente de la transformada wavelet Haar en la GPU, así como la visualización interactiva de la señal de metilación desde los resultados de la transformada ubicados en la memoria de la GPU. Este enfoque permite la visualización e identificación precisa de DMRs, ofreciendo una herramienta flexible y amigable con una carga computacional extremadamente baja. La contribución de este *paper* sobre el trabajo previo realizado es, por un lado, verificar que el software permite la visualización interactiva de diferentes señales de metilación a diferentes niveles de resolución. Por otro lado, verificar que el tiempo de ejecución requerido por este enfoque es significativamente menor que otras propuestas, dada la intrínseca estructura paralela de la GPU. De hecho, los resultados de la evaluación muestran que la implementación el GPU propuesta requiere tiempos de ejecución dos órdenes de magnitud menores que las herramientas existentes.

El resto del trabajo se organiza como sigue: La sección II describe algunas herramientas existentes para visualización de los datos de metilación. La sección III muestra los requerimientos de la paralelización de GPU para la transformada wavelet Haar, así como la visualización de la señal desde los datos ubicados en la memoria de la GPU. La sección IV muestra la evaluación de la implementación propuesta. Finalmente, la sección V muestra algunas conclusiones relevantes y el trabajo futuro a realizar.

II. Antecedentes

En esta sección describimos brevemente las propuestas existentes hasta el momento, según nuestro conocimiento. Compararemos los resultados del análisis de metilación de diferentes muestras y cómo identifican DMRs.

BSmooth [10] se desarrolló en 2012 siendo la primera herramienta para la identificación de DMRs teniendo en cuenta la variabilidad biológica. También tiene en cuenta la cobertura (número de muestras alineadas en cada localización del ADN) o la calidad de las muestras si la cobertura es baja. Proporciona las regiones del ADN que muestran diferencias de metilación consistentes entre grupos de muestras. BSmooth asume que el perfil de metilación del genoma de referencia varía suavemente. Por lo tanto, esta herramienta ofrece una estimación suave.

Biseq es una herramienta diseñada para la identificación de DMRs en zonas concretas del ADN controladas por el ratio de falso descubrimiento (FDR - false discovery ratio) [11]. Este objetivo lo alcanza priorizando aquellas áreas de mayor cobertura. MOABS es otra herramienta para la identificación de DMRs [12]. Con el objetivo de aumentar la capacidad estadística, asume que los CpG vecinos tiene niveles similares de metilación. Sin embargo, este análisis promediado falla en zonas con baja densidad en CpGs. Además, utiliza la métrica *P value* con la que indica las DMRs pero no su magnitud, en el sentido en que no discrimina regiones con diferencias pequeñas de regiones con diferencias significativas. Existe otra herramienta que utilizan modelos Bayesianos para la identificación de posiciones con diferencias de metilación [17]. Esta herramienta modela la variabilidad biológica a partir de la distribución beta.

Methylsig es un paquete de análisis estadístico para analizar diferencias de metilación entre muestras de diferentes grupos de tratamientos o enfermedades [18]. Tiene en cuenta tanto la cobertura como la variabilidad biológica, a partir de una aproximación beta-binomial de las muestras de cada grupo, en cada posición CpG. Sin embargo, la visualización de los resultados obtenidos se debe procesar con el paquete de programación R. DSS-single [13] es otra herramienta reciente que utiliza técnicas estadísticas. Esta herramienta utiliza técnicas de suavizado sobre la información de metilación de las posiciones contiguas para estimar la señal de metilación. En este sentido, puede identificar regiones más amplias que las propias muestras. Otras herramientas matemáticas, utilizadas en diferentes propuestas, hacen uso de modelos ocultos de Markov (HMM), como Bisulfighter [19], o aplicación del principio de entropía de Shannon en QDMR [20].

También, encontramos una propuesta basada en mezclas de modelos basados en wavelets para la identificación de posiciones diferencialmente metiladas [21]. Se utiliza una mezcla de funciones wavelet para adaptar la importancia de las posiciones contiguas metiladas en la estimación del nivel de metilación de cada posición. Sin embargo, no utiliza la transformada wavelet para el procesamiento de la señal de metilación, tal y como realiza nuestra propuesta. En su lugar, utiliza aproximaciones Bayesianas en la identificación de DMRs.

Finalmente, incluso existen trabajos que hacen un estudio recopilatorio de las propuestas existentes hasta la fecha [14].

III. Visualización de los datos de metilación desde la GPU

Para modelar el perfil de metilación por muestra proponemos un enfoque basado en la aplicación de transformaciones ortogonales o bi-ortogonales que codifican adecuadamente la información de metilación en el dominio tiempo-frecuencia [16]. Utilizamos la transformada wavelet Haar [22], [23] dada su idoneidad para las transiciones bruscas de nivel entre posiciones contiguas, lo que resulta especialmente útil en la detección de bases metiladas, tanto aisladas como agrupadas en islas CpG.

La Figura 1 muestra el flujo típico de análisis

de metilación del ADN así como el tipo de resultado obtenido. El proceso comienza con los ficheros tipo FastQ obtenidos de las máquinas de secuenciación [3], [5], [6]. Herramientas software de análisis de metilación como la amplimente utilizada Bismark [7] o la más reciente y eficiente HPG-Methyl [8], [9] proporcionan información a nivel de base del alineamiento y estado de metilación de cada secuencia de entrada (o *read*), ofreciendo los resultados en ficheros con formato SAM o BAM. En este punto, herramientas con HPG-Hmapper [25], [**?**] pueden recoger los datos en ficheros BAM para construir los mapas de metilación. El resultado se almacena en ficheros CSV, tal y como se puede observar en la Figura 1. Estos ficheros CSV incluyen las posiciónes del ADN con información de metilación y el número de *reads* que se han encontrado en esa posición.

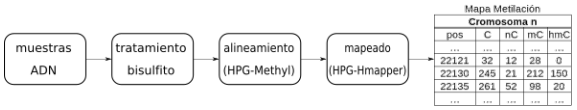


Fig. 1. Procedimiento típico de análisis de metilación.

A. Construcción de la señal de metilación

A partir de los ficheros CSV se puede obtener un mapa personalizado de metilación donde aparezcan todas y cada una de las posiciones del ADN de una persona que contienen una C metilada, de tal manera que se pueda comparar con mapas de otras personas, tejidos o muestras e identificar regiones diferencialmente metiladas (DMRs). Para alcanzar este último objetivo la idea es construir una señal de metilación donde tengamos un valor de metilación para cada posición del ADN. Por ello, el primer paso consiste en construir la señal de metilación desde los ficheros CSV y que se puedan visualizar a diferentes niveles de resolución. Para cada posición del ADN construimos la señal del siguiente modo: sea m_j el número de *reads* encontradas con C metilada en la posiciń j. Sea u_j el número de *reads* encontradas con C no metilada en la posición j. Las *reads* con una base diferente de C, ya sea metilada o no, en la posición j, se descartarán [10], [13]. La cobertura n_j para la posición j se calcula como $n_j = m_j + u_j$. De este modo, cada elemento v_j de la señal de metilación V, se calcula como

$$v_j = \frac{m_j}{u_j} \tag{1}$$

Como se puede deducir, el rango de v_j se halla entre 0 y 1. Con objeto de realizar una evaluación robusta, se han usado estructuras de datos flexibles que proporcionan el almacenamiento simultaneo de datos provenientes de diferentes muestras. Concretamente, se utilizado el conjunto de datos que ofrece BSseqData para otras propuestas [10], [13]. Este conjunto de datos contiene 6 muestras, de diferentes pacientes, para cada posición del ADN. El formato del fichero en este conjunto de datos es en una matriz de 380000 filas (correspondientes a las posiciones metiladas del ADN en el cromosoma 21) por 13 columnas. Para cada fila j, la primera columna contiene la posición en el cromosoma, las siguientes seis columnas contienen la cobertura total n_j de cada muestra y las últimas seis columnas contienen la cobertura de metilación m_j de cada muestra, para un total de trece valores por fila.

Nuestra implementación lee este tipo de fichero y almacena los datos leidos en una matriz dinámica extendida, conteniendo una columna por cada posición del ADN (la señal de metilación debe tener en cuenta todas las posiciones correlativas, no solo las metiladas). La matriz tiene tantas filas como número de muestras. Cada elemento (i, j) de la matriz, contiene el valor de metilación en la posición j de la muestra i. La posición que no tenga correspondencia con el fichero del conjunto de datos tendrá asignado un valor de cero. La Figura 2 muestra el código correspondiente a la creación de la matriz extendida en la memoria compartida de la CPU.

```
// allocate memory for the extended matrix of data
mc_full    = new float*[samples];
mc_full[0] = new float [samples * dimension];

// address assignment for each sample start
for (int i = 1; i < samples; i++)
    mc_full[i] = mc_full[i - 1] + dimension;

// fill the matrix
for (int m = 0; m < samples; m++)
{
    // fill with zeros the data matrix
    for (int n = 0; n < dimension + 1; ++n)
        mc_full[m][n] = 0.0;
    // fill the methylated positions with data
    // from the input file
    for (uint k = 0; k < data.size(); k++)
        mc_full[m][data[k][0] - lower_limit]
                                    = mc[k][m];
}
```

Fig. 2. Creación de la matriz extendida en la CPU.

B. Transformación wavelet Haar de la señal de metilación

Una vez se ha construido la señal de metilación, se aplica la transformada wavelet discreta Haar iterativamente hasta que se alcanza el nivel de resolución deseado. En cada interación, la transformada Haar procesa pares de posiciones adyacentes de la parte de escalado del nivel anterior $n-1$ (o la señal original si está procesando el nivel $n = 1$), reduciendo en cada nivel la longitud de la señal a la mitad del nivel anterior. Concretamente, se aplica una función (o filtro paso-bajo) que devuelve la señal escalada como los valores promedio de pares de valores adyacentes. También se podría computar los coeficientes wavelet de cada nivel como la diferencia entre los pares de valores adyacentes, pero para la visualización de la señal de metilación y para la identificación de DMRs solo se precisa la

parte escalada, mientras que los coeficientes wavelet no se necesitan.

Por tanto, por cada pareja de valores adyacentes v_j^{n-1} y v_{j+1}^{n-1} de la señal de la iteración $n-1$, la transformada wavelet Haar calcula cada elemento v_j^n de una versión con menor resolución de la señal, así como los correspondientes coeficientes wavelet d_j^n, como se puede ver en las ecuaciones 2 y 3. Se debe tener en cuenta que, para $n=0$, $v_j^0 = v_j$ (los valores descritos en la ecuación 1).

$$v_j^n = \frac{v_j^{n-1} + v_{j+1}^{n-1}}{\sqrt{2}} \quad (2)$$

$$d_j^n = \frac{v_j^{n-1} - v_{j+1}^{n-1}}{\sqrt{2}} \quad (3)$$

Comenzando desde la versión de resolución más baja de la señal y utilizando los coeficientes wavelet, la señal original de metilación se puede reconstruir completa, aplicando la inversa de la transformada wavelet. Como lo que se desea es visualizar la señal desde cualquiera de los niveles de resolucin, la reconstrucción se realiza a partir de la misma inversa de la transformada wavelet, pero con los coeficientes wavelet con valor cero. Como ejemplo ilustrativo, la Figura 3 muestra una señal de matilación recostruida con el enfoque propuesto. La señal original se sitúa en la parte inferior de la figura. Está construida desde un segmento de ADN de 1600 nucleótidos. El resto de las gráficas (comenzando desde la situada encima de la señal original y en direcciń ascendente) muestran la señal de metilación recontruida para las iteraciones dos, cuatro, seis, ocho y diez de la transformada wavelet Haar. Todas las gráficas muestran en el eje X la posición dentro del cromosoma humano de referencia. En el eje Y se muestra los valores de v_j^n.

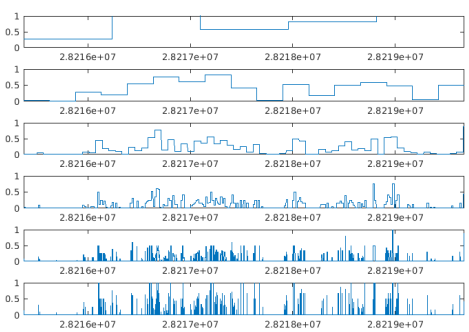


Fig. 3. Señales de metilación con diferentes niveles de resolución obtenidas con el enfoque propuesto.

C. *Implentación de la transformada wavelet Haar en GPU*

El proceso descrito en el apartado anterior se realizó en CPU utilizando Matlab [16]. Sin embargo, la capacidad de paralelismo y computaciń de las tarjetas gráficas actuales pueden acelerar significativamente la aplicación de la transformada wavelet a la señal de metilaciń, calculando los coeficientes del filtro paso-bajo (o escalado) de la transformada wavelet. Nuestra aplicación trabaja enteramente en la memoria de la GPU, minimizando la transferencia de datos entre la memoria de la CPU y la memoria de la GPU.

El primer paso es la asignación de espacio en la memoria compartida de la GPU para almacenar la matrix de datos extendida. Utilizamos la función *cudaMallocPitch* para este propósito, que devuelve , la longitud asignada a cada fila de la matriz. El tamaño de cada fila se calcula como un múltiplo del tamaño del *warp* (32 en nuestro caso). El siguiente paso copia la matriz extendida desde la memoria de la CPU al espacio de memoria de la GPU reservado en el paso previo. Para ello se hace uso de la función *cudaMemcpy2D*. Este paso se realiza una sola vez y ningún dato de metilación más se transfiere de la CPU a GPU durante el proceso de visualización.

Los siguientes pasos tienen que ver con los *kernels* de CUDA para ejecutar la transformación wavelet sobre los datos almacenados en la matriz extendida. Respecto a estos *kernels*, se debe tener en cuenta que la dependencia entre los datos, en la transformación wavelet, precisa de sincrinización global entre los bloques, además de la sincronización a nivel de bloque ofrecida por la función *__syncthreads()*. Como la cantidad de datos a transformar es mucho mayor que el tamaño de cada bloque, debemos utilizar paralelización dinámica. Esta característica proporciona a los *kernels* de CUDA la capacidad de invocar a *kernels* hijo y consumir los datos provenientes de estos hijos una vez han terminado el trabajo con la totalidad de los datos. Sin embargo, el hecho de que cada *kernel* hijo solo devuelva los datos procesados a padre cuando los hilos de todos los bloques han terminado puede representar un problema de sincronización. Este problema lo hemos resuelto implementando el proceso mostrado en la Figura 4, que consiste en los siguientes pasos: primero, el *host* invoca la *kernel* principal con tantos hilos como muestras a transformar, indicando los límites del segmento de ADN sobre el que se va a aplicar la transformada wavelet, así como el nivel de transformación deseado. Segundo, cada hilo (responsable de cada una de las muestras) invoca un *kernel* hijo que realiza una copia del segmento de ADN a transfomar en un *buffer* auxiliar. Tras ello, invoca a otr *kernel* hijo para aplicar la transformada wavelet Haar. A continuación invoca a un tercer *kernel* hijo para copiar los resultados obtenidos en el hijo anterior en la primera mitad del *buffer* auxiliar, dejando los resultado accesibles para el siguiente nivel de transformación. Tercero, los dos pasos últimos se repiten tantas veces como niveles de resolución se ha establecido en la invocación del *host*. Finalmente, el *kernel* principal devuelve el control al *host* con los resultados de la transformaciń wavelet disponibles en el *buffer* auxiliar. Es importante resaltar que los datos han permanecido en la memoria del dispositivo durante

todo el proceso, evitando la transferencia de datos entre la GPU y la CPU.

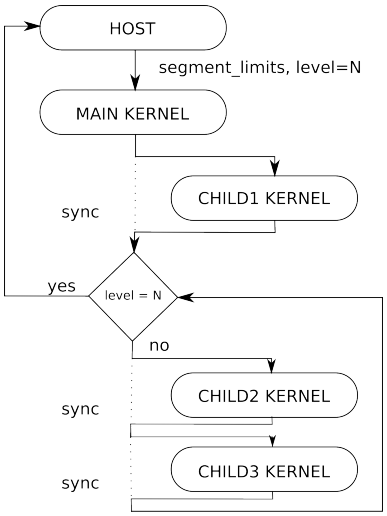


Fig. 4. Proceso de paralelización.

D. Visualización de las señales de metilación

Una vez se ha calculado la transformada wavelet Haar sobre el segmento de señal deseado, el resultado se halla disponible en la memoria global de la GPU. Por tanto, estos resultados se pueden renderizar directamente utilizando la librería `OpenGL`. La renderización de la señal requiere el establecimiento de un vínculo entre el contexto de `OpenGL` y la localización de memoria donde se hallan almacenados los datos, el ajuste de los datos a renderizar a la ventana donde trabaja `OpenGL` y el renderizado final de los datos. Por lo tanto, hemos creado una nueva clase conteniendo los métodos básicos *InitializeGL()* y *PaintGL()*, además de aquellos necesarios para registrar y mapear un *Vertex Buffer Object* (VBO) en los datos generados por CUDA. El método *InitializeGL()* inicializa las funciones `OpenGL`, genera un VBO, entrablece el vínculo de este VBO con un *array* donde se almacenan los puntos a renderizar, le asigna un tamaño inicial y lo registra en un manejador (*handler*) de CUDA. El método *PaintGL()* renderiza en cada frame los datos procesados por CUDA. Como la cantidad de datos a renderizar es diferente en cada llamada, el proceso se compone de los siguientes pasos: primero, el número de muestras a renderizar así como el número de datos por muestra se asignan a cada atriburo de la clase. Entonces el manejador de CUDA se mapea hacia el puntero del *array* con los datos de la señal, CUDA calcula la transformación wavelet y el resultado se almacena en el *array* VBO para ser renderizados por el método *PaintGL()*.

IV. Evaluación de prestaciones

En esta sección mostramos la evaluación de prestaciones de la implementación descrita en la sección III-C. Como el propósito de nuestra investigación es reducir el tiempo entre la secuenciación de los datos genómicos y su interpretación por parte de los investigadores biomédicos, hemos establecido el tiempo de ejecución como la principal métrica de rendimiento. Con propósito comparativo, hemos reproducido los resultados obtenidos con las herramientas BSmooth [10] y DSS-single [13] utilizando el conjunto de datos BsseqData para computar los datos de metilación e identificar los DMRs. Este conjunto de datos contiene datos de metilación de los cromosomas 21 y 22 de 6 muestras (pacientes) diferentes, tres de ellas con cáncer y las otras tres como muestras de control. El archivo contiene 958.098 posiciones metiladas, cubriendo un total de 73.902.598 posiciones del ADN en cada una de las seis muestras. Hemos computado la transformada wavelet de la señal de metilación utilizando tanto la herramienta Matlab (Version 18.0) como la implementación sobre GPU propuesta, con el propósito de comparar las prestaciones. Con objeto de garantizar un estudio del peor caso, las medidas mostradas en esta sección para la implementación propuesta incluyen tanto el tiempo de computación de la transformada wavelet como la visualización interactiva de la señal resultante. Por el contrario, los tiempos de ejecución mostrados para la herramientas existentes incluyen exclusivamente el tiempo necesario para identificar los DMRs por los principales métodos estadísticos (sus resultados se visualizan aparte, utilizando otros comandos de R).

TABLA I
Tiempos de ejecución necesarios para la identificación de DMRs con diferentes herramientas.

Herramienta	T. ejec. (s)	GPU mem.
BSmooth	47.3	-
DSS-single	449.4	-
Matlab	35.1	-
HPG-Dhunter	0.154	3864 MiB

Hemos utilizado un ordenador basado en un procesador Intel(R) Xeon(R) CPU E5-2650 v4 @ 2.20 GHz, con 12 *cores* y 2 hilos de procesamiento por *core*, 64 Gb de memoria RAM y una tarjeta gráfica GeForce GTX 1080 con 2560 *cores* y 8 Gb de memoria RAM GDDR5X. El algoritmo de la transformada wavelte Haar se ha implementado con C++ y se ha utilizado la librería CUDA 9.2.88.

La Tabla I muestra el tiempo requerido para procesar los datos de metilación de diferentes herramientas. Las dos primeras filas muestran el tiempo de procesado por las herramientas existentes (BSmooth utiliza 12 *cores*). La tercera fila muestra el tiempo de ejecuciń para la propuesta de wavelet utilizando múltiples *cores* (el máximo proporcionado por Matlab). La última fila muestra el tiempo de procesamiento de la implementación en GPU propuesta. En el último caso, la tabla también muestra el total de memoria necesario de GPU para realizar el cálculo y la visualización de las señales. De este modo, el lector puede ver el mínimo de memoria

necesario para implementar la propuesta de GPU con otras tarjetas gráficas.

Cuando comparamos las tres primeras filas de la Tabla I, se puede observar que la implementación con Matlab necesita un tiempo de ejecución menor que BSmooth y ambos son un orden de magnitud menores que el requerido por DSS-single. La razón de este comportamiento es que las implementaciones tanto de Matlab como de BSmooth, permiten la ejecución paralela, de modo que hemos testeado estas implementaciones con 12 *cores*. Sin embargo, DSS-single solo permite la ejecución secuencial en un solo hilo. Por otro lado, el tiempo requerido para la ejecución en GPU es dos órdenes de magnitud inferior al necesario por la implementación con Matlab. Este rendimiento se puede alcanzar debido a que el total de la matriz de datos (necesitando 3864 MiB) se pueden almacenar en la memoria de la GPU (de 8 Gb). Además, no existe transferencia de datos entre la memoria del *host* y de la GPU ni en la fase de transformación wavelet ni en la de visualización de la señal.

TABLA II
TIEMPOS DE EJECUCIÓN PARA CONJUNTOS DE DATOS DE DIFERENTES LONGITUDES.

Herramienta	Doble T. ejec. (s)	GPU mem.
Matlab	62.08	-
HPG-Dhunter	0.203	4999 MiB
	Cuádruple	
	T. ejec. (s)	**GPU mem.**
Matlab	108.1	-
HPG-Dhunter	0.204	4997 MiB

Tras ello, hemos estudiado las prestaciones de la implementación propuesta con conjuntos de datos más largos, con objeto proporcionar al lector una idea del nḿero de cromosomas y/o muestras que se pueden procesar simultaneamente con este enfoque y la tarjeta gráfica referida. Para cargas de trabajo que excedan el tamaño de memoria de la tarjeta gráfica puede ser necesaria la segmentación de los datos o reducir las ventana de visualización de los resultados (un solo cromosoma o parte del mismo). Sin embargo los resultados mostrados en este trabajo indican que el enfoque e implementación propuestos ofrecen resultados significativamente mejores en cuanto a tiempos de ejecución que las soluciones existentes, que además no pueden ofrecer diferentes niveles de resolución ni aprovechar el paralelismo que ofrecen las tarjetas gráficas. En este caso particular, hemos extendido el archivo del conjunto de datos original (compuesto por seis muestras de ADN, cada una con 958.541 posiciones metiladas cubriendo 73.933.448 posiciones de ADN y precisando 3864 MiB de memoria GPU) duplicando su longitud para algunas muestras (copiando y concatenando cada muestra sobre sí misma y reasignando posiciones). La Tabla II muestra el resultado para diferentes tamaños de muestras. Las filas bajo el epígrafe de *Doble* muestras los resultados para un conjunto de datos con cuatro muestras (el conjunto original tenía seis muestras) cuya longitud es el doble de la longitud original. Las filas bajo el epígrafe de *Cuádruple* muestran los resultados para un conjunto de datos con dos muestras cuya longitud es cuatro veces superior a la longitud original. En ambos casos, la matriz extendida (requiere menos de 5Gb de memoria) se puede almacenar completamente en la memoria de la GPU, cuyo tamaño es 8 Gb.

La Tabla II muestra que los tiempos de ejecución necesarios para la implementación propuesta con GPU apenas varían a pesar de las diferencias de longitud, gracias a que las matrices pueden almacenarse en la memoria de la GPU. Estos tiempos son tres órdenes de magnitud inferiores a los necesarios por la implementación con Matlab en la CPU. De este modo, podemos utilizar una tarjeta gráfica como la que estamos usando en este experimento para, en función del tamaño total de los cromosomas humanos, procesar hasta seis muestras de los cromosomas 16 a 22, además del cromosoma Y. O hasta cuatro muestras de los cromosomas 7 al 22, además de los cromosomas X e Y. O, también, hasta dos muestras de cualquier cromosoma humano.

Finalmente, también hemos realizado un estudio comparativo de la visualización de resultados ofrecidos por cada enfoque. Con objeto de realizar una comparaciń justa, hemos utilizado el mismo conjunto de datos utilizado en esos trabajos, los datos de Hansen [26], consistentes en datos provenientes de tres pares de muestras de colon tumor/normal secuenciadas en una máquina secuenciadora ABI SOLiD según el Whole-Genome Bisulfite Sequencing (WGBS).

La Figura 5 muestra en resultado del anĺisis realizado con Bsseq en un segmento de 2682 nucleótidos correspondiente al cromosoma 21, desde la posición 28.337.776 hasta la posición 28.340.457, para un total de 5000 posiciones de ADN. Esta Figura muestra en el eje X las posiciones en el ADN y en el eje Y el perfil de metilación entre 0 y 1. Muestra seis gráficas, tres de ellas corresponden a las curvas de metilación de células tumorales y las otras 3 muestran las curvas de metilación de células normales. La región sombreada en el centro de la gráfica muestra una región diferencialmente metilada detectada por este software.

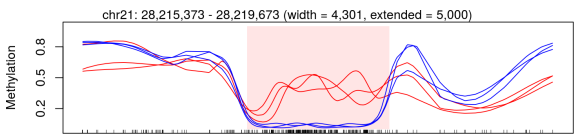


Fig. 5. Señales de metilación ofrecidas por la herramienta Bsseq.

La Figura 6 muestra el resultado del análisis realizado por el método estadístico DSS-single [13] sobre los datos de Hansen. Sin embargo, con objeto se proporcionar más claridad a los gráficos,

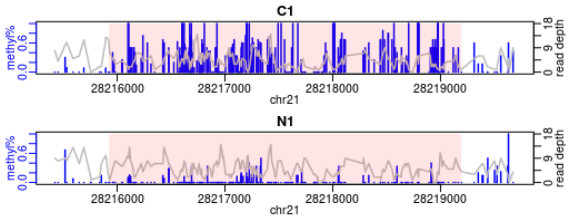


Fig. 6. Señales de metilación ofrecidas por la herramienta DSS-single.

se han separado las gráficas de los resultados del análisis para cada una de las muestras. Concretamente, esta Figura muestra solo dos gráficas, una correspondiente a una célula cancerosa (etiquetada como C1) y otra correspondiente a una célula normal (etiquetada como N1).

Cada gráfica de la Figura 6 muestra dos trazos. El trazo azul (la señal con múltiples picos) muestra los datos de metilación, mientras que la señal gris (el trazo de línea más continua) muestra la profundidad de las *reads* o cobertura en cada posición (número de muestras alineadas en cada posición). De nuevo, la zona sombreada representa la región dondes este método ha identificado niveles diferencialmente significativos. También, las señales de metilación son diferentes debido a la técnica de suavizado utilizada en la herramienta Bsseq. Se debe tener en cuenta que las gráficas de las Figuras 5 y 6 se han obtenido con comandos específicos para este cometido desde el entorno R.

La Figura 7 muestra las gráficas correspondientes a las señales de metilación obtenidas con el método propuesto. Cada gráfica en esta Figura muestra las señales de metilación para las mismas muestras celulares que las señales mostradas en la Figura 6. En este caso, cada gráfica muestra dos trazos: la señal azul con múltiples picos, cuyos valores en el eje Y oscilan entre 0 y 1, es la señal V de metilación original, construida como se describe en la sección III. Además, hay otra señal escalonada correspondiente al noveno nivel de la transformada inversa Haar aplicada a los coeficientes wavelet. Se puede ver claramente que este enfoque permite detectar visualmente las regiones con diferencias de metilación significativas (las diferencias significativas la gráfica superior e inferior se pueden apreciar en la Figura entre las posiciones 1000 y 3500, correspondientes a las posiciones 28.338.776-28.341.276 del ADN). Con objeto de no perjudicar la claridad de visualización de las gráficas, las señales correspondientes a la inversa de la transformada wavelet Haar en otros niveles se han omitido, si bien muestran la existencia de esta región diferencialmente metilada con claridad.

Comparando las Figuras 6 y 7, se puede ver claramente que el método propuesto ofrece señales donde es muy sencillo detectar regiones diferencialmente metiladas, de un modo similar a como las encuentran los métodos existentes. Sin embargo, el método propuesto ofrece estas

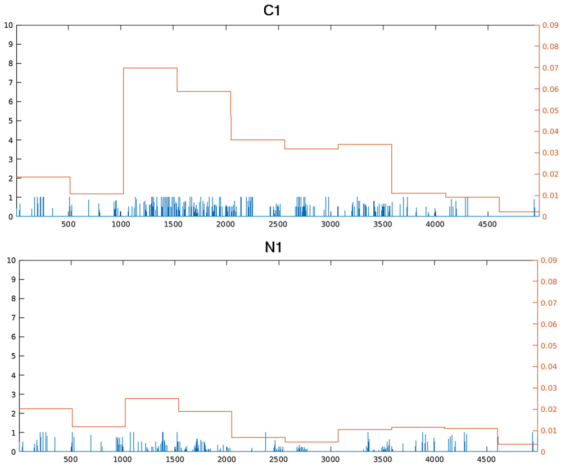


Fig. 7. Señales de metilación ofrecidas por el método propuesto.

regiones con tiempos de ejecución mucho más reducidos. Se pueden obtener señales (por lo tanto, permite detectar regiones) con diferentes niveles de resolución, mientras que las propuestas existentes obtienen regiones con un nivel fijo de resolución.

V. Conclusiones y trabajo futuro

En este trabajo hemos propuesto la transformación de los resultados del análisis de la metilación del ADN en una señal de metilación, la paralelización de la transformada wavelet Haar en GPU, así como la visualización de los resultados de transformación desde la misma GPU. Este enfoque permite, por primera vez, la interacción visual de diferentes señales de metilación con diferentes niveles de resolución. Este enfoque se puede utilizar para la detección visual precisa de DMRs desde una herramienta flexible y amigable, y con un tiempo de ejcución interactivo.

Los resultados de evaluación obtenidos por el enfoque propuesto muestran que los tiempos de ejecución para el proceso completo hasta lograr la visualización de la señal de metilación es varios órdenes de magnitud inferior a las herramientas existentes, debido al inherente paralelismo de la GPU. Gracias a los reducidos tiempos de ejecución, las señales de metilación pueden renderizarse interactivamente. Por lo tanto, el enfoque propuesto permite al usuario encontrar visualmente regiones diferencialmente metiladas, al contrario que las herramientas existentes.

Como trabajo futuro a realizar, planeamos el desarrollo de una nueva herramienta para la identificación de DMRs por lotes. De este modo, se dispondría de un archivo con todas las DMRs de los cromosomas seleccionados y entre todas las muestras seleccionadas, sin depender de la capacidad total de memoria de la tarjeta gráfica. Con este fichero se podría localizar con mayor eficiencia las DMRs en el enfoque propuesto, para analizar exactamente las regiones de interés identificadas.

Agradecimientos

Este trabajo ha sido financiado por el MCIU y los fondos FEDER de la Comisión Europea bajo el proyecto del Plan Nacional de referencia RTI2018-098156-B-C55.

Referencias

[1] VD de Mello, L Pulkkinen, M Lalli, M Kolehmainen, J Pihlajamaki, and M. Uusitupa, "DNA methylation in obesity and type 2 diabetes.," *Annals of Medicine*, vol. 46, no. 3, pp. 103–13, 2014.

[2] Alexander Raciti, Cecilia Nigro, Michele Longo, Luca Parrillo, Claudia Miele, Pietro Formisano, and Francesco Béguino, "Personalized medicine and type 2 diabetes: lesson from epigenetics.," *Epigenomics*, vol. 6, no. 2, pp. 229–238, 2014.

[3] Peter W. Laird, "Principles and challenges of genome-wide dna methylation analysis," *Nature Reviews Genetics*, vol. 11, pp. 191–203, 2010.

[4] Miao Yu, Gary Cc Hon, Keith E. Szulwach, Chun-Xiao Song, Peng Jin, Bing Ren, and Chuan He, "TET-assisted bisulfite sequencing of 5-hydroxymethylcytosine," *Nature Protocols*, vol. 7, no. 12, pp. 2159–2170, 2012.

[5] Miao Yu, Gary Cc Hon, Keith E. Szulwach, Chun-Xiao Song, Liang Zhang, Audrey Kim, Xuekun Li, Qing Dai, Beomseok Park, Jung-Hyun Min, Peng Jin, Bing, and Chuan He, "Base-resolution analysis of 5-hydroxymethylcytosine in the mammalian genome," *Cell*, vol. 149, no. 6, pp. 1368–1380, 2012.

[6] Zongli Xu, Jack A. Taylor, Yuet-Kin Leung, Shuk-Mei Ho, and Liang Niu, "oxBS-MLE: an efficient method to estimate 5-methylcytosine and 5-hydroxymethylcytosine in paired bisulfite and oxidative bisulfite treated dna," *Bioinformatics*, vol. 32, no. 23, pp. 3667–3669, 2016.

[7] Felix Krueger and Simon R. Andrews, "Bismark: a flexible aligner and methylation caller for bisulfite-seq applications," *Bioinformatics*, vol. 27, no. 11, pp. 1571–1572, 2011.

[8] Joaquín Tárraga, Mariano Pérez, Juan M. Orduña, José Duato, Ignacio Medina, and Joaquín Dopazo, "A parallel and sensitive software tool for methylation analysis on multicore platforms," *Bioinformatics*, vol. 31, no. 19, pp. 3130, 2015.

[9] Ricardo Olanda, Mariano Pérez, Juan M. Orduña, Joaquín Tárraga, and Joaquín Dopazo, "A new parallel pipeline for DNA methylation analysis of long reads datasets," *BMC Bioinformatics*, vol. 18, no. 1, pp. 161, 2017.

[10] Kasper D. Hansen, Benjamin Langmead, and Rafael A. Irizarry, "Bsmooth: from whole genome bisulfite sequencing reads to differentially methylated regions," *Genome Biology*, vol. 13, no. 10, pp. R83, Oct 2012.

[11] Katja Hebestreit, Martin Dugas, and Hans-Ulrich Klein, "Detection of significantly differentially methylated regions in targeted bisulfite sequencing data," *Bioinformatics*, vol. 29, no. 13, pp. 1647–1653, 2013.

[12] Deqiang Sun, Yuanxin Xi, Benjamin Rodriguez, Hyun Jung Park, Pan Tong, Mira Meong, Margaret A. Goodell, and Wei Li, "Moabs: model based analysis of bisulfite sequencing data," *Genome Biology*, vol. 15, no. 2, pp. R38, Feb 2014.

[13] Hao Wu, Tianlei Xu, Hao Feng, Li Chen, Ben Li, Bing Yao, Zhaohui Qin, Peng Jin, and Karen N. Conneely, "Detection of differentially methylated regions from whole-genome bisulfite sequencing data without replicates," *Nucleic Acids Research*, vol. 43, no. 21, pp. e141, 2015.

[14] Adib Shafi, Cristina Mitrea, Tin Nguyen, , and Sorin Draghici, "A survey of the approaches for identifying differential methylation using bisulfite sequencing data," *Briefings in Bioinformatics*, vol. 19, no. 5, pp. 737–753, 03 2018.

[15] Aimin Chen, Shuk-Mei Ho, Yuet-Kin Leung, Changchun Xie, Ding-Xin Long, and Catherine Hoyo, "Differential methylation values in differential methylation analysis," *Bioinformatics*, vol. 35, no. 7, pp. 1094–1097, 09 2019.

[16] Lisardo Fernández, Luis Ordu na, Mariano Pérez, and Juan M. Ordu na, "A new approach for the visualization of dna methylation results," in *Proceedings of the 18th International Conference on Mathematical Methods in Science and Engineering*, 2018.

[17] Hao Feng, Karen N. Conneely, and Hao Wu, "A bayesian hierarchical model to detect differentially methylated loci from single nucleotide resolution sequencing data," *Nucleic Acids Research*, vol. 42, no. 8, pp. e69, 2014.

[18] Yongseok Park, Maria E. Figueroa, Laura S. Rozek, and Maureen A. Sartor, "Methylsig: a whole genome dna methylation analysis pipeline," *Bioinformatics*, vol. 30, no. 17, pp. 2414–2422, 2014.

[19] Yutaka Saito, Junko Tsuji, and Toutai Mituyama, "Bisulfighter: accurate detection of methylated cytosines and differentially methylated regions," *Nucleic Acids Research*, vol. 42, no. 6, pp. e45, 2014.

[20] Yan Zhang, Hongbo Liu, Jie Lv, Xue Xiao, Jiang Zhu, Xiaojuan Liu, Jianzhong Su, Xia Li, Qiong Wu, Fang Wang, and Ying Cui, "Qdmr: a quantitative method for identification of differentially methylated regions by entropy," *Nucleic Acids Res*, vol. 39, pp. e58, 2011.

[21] Wonyul Lee and Jeffrey S. Morris, "Identification of differentially methylated loci using wavelet-based functional mixed models," *Bioinformatics*, vol. 32, no. 5, pp. 664–672, 2016.

[22] Radomir S. Stankovic and Bogdan J. Falkowski, "The haar wavelet transform: its status and achievements.," *Computers & Electrical Engineering*, vol. 29, no. 1, pp. 25–44, 2003.

[23] V. Ashok, T. Balakumaran, C. Gowrishankar, I. L. A. Vennila, and A. Nirmal Kumar, "The fast haar wavelet transform for signal & image processing," *International Journal of Computer Science and Information Security*, vol. abs/1002.2184, 2010.

[24] Yuanxin Xi, Christoph Bock, Fabian Muller, Deqiang Sun, Alexander Meissner, and Wei Li, "RRBSMAP: a fast, accurate and user-friendly alignment tool for reduced representation bisulfite sequencing.," *Bioinformatics*, vol. 28, no. 3, pp. 430–432, 2012.

[25] César González, Mariano Pérez, Juan M. Ordu na, Javier Chaves, and Ana-Bárbara García, "On the use of binary trees for dna hydroxymethylation analysis," in *5th International Workshop on Parallelism in Bioinformatics, as part of ICA3PP 2017*, 2017.

[26] Kasper D. Hansen, W. Timp, H.C. Bravo, and Benjamin Langmead, "Increased methylation variation in epigenetic domains across cancer types," *Nature genetics*, vol. 43, no. 8, pp. 768–775, 2011.

[27] NVIDIA, "Dynamic parallelism in cuda," 2018.

[28] NVIDIA, "Nvidia cuda programming guide 9.2," 2018.

[29] G.P. Nason, *Wavelet Methods in Statistics with R*, Springer Publishing Company, Incorporated, 1 edition, 2008.

[30] Clive Loader, *Local regression and likelihood*, New York: Springer-Verlag, 1999.

[31] Ke Chen, Jing Zhang, Zhongqiang Guo, Qin Ma, Zhengzheng Xu, Yuanyuan Zhou, Ziying Xu, Zhongwu Li, Yiqiang Liu, Xiongjun Ye, Xuesong Li, Bifeng Yuan, Yuwen Ke, Chuan He, Liqun Zhou, Jiang Liu, and Weimin Ci1, "Loss of 5-hydroxymethylcytosine is linked to gene body hypermethylation in kidney cancer," *Cell Research*, vol. 26, no. 1, pp. 103–118, 2016.

[32] Lu Wen, Xianlong Li, Liying Yan, Yuexi Tan, Rong Li, Yangyu Zhao, Yan Wang, Jingcheng Xie, Chuan He, Ruiqiang Li, Fuchou Tang, and Jie Qiao, "Whole-genome analysis of 5-hydroxymethylcytosine and 5-methylcytosine at base resolution in the human brain," *Genome Biology*, vol. 15, no. 3, pp. R49, 2014.

[33] Michael J. Booth, Miguel R. Branco, Gabriella Ficz, David Oxley, Felix Krueger, Wolf Reik, and Shankar Balasubramanian, "Quantitative sequencing of 5-methylcytosine and 5-hydroxymethylcytosine at single-base resolution," *Science*, vol. 336, no. 6083, pp. 934–937, 2012.

[34] Michael J. Booth, Tobias W. Ost, Dario Beraldi, Neil M Bell, Miguel R. Branco, Wolf Reik, and Shankar Balasubramanian, "Oxidative bisulfite sequencing of 5-methylcytosine and 5-hydroxymethylcytosine," *Nature Protocols*, vol. 8, pp. 1841–1851, 2013.

[35] A W Drong and McCarthy M. Lindgren, C.M. and, "The genetic and epigenetic basis of type 2 diabetes and obesity.," *Clin Pharmacol Ther*, vol. 92, no. 6, pp. 707–15, 2012.

[36] Cecile Haumaitre, "Epigenetic regulation of pancreatic islets," *Current Diabetes Reports*, vol. 13, no. 5, pp. 624–632, 2013.

[37] Stephanie-May Ruchat, Marie-France Hivert, and Luigi

Bouchard, "Epigenetic programming of obesity and diabetes by in utero exposure to gestational diabetes mellitus.," *Nutr Rev*, vol. 71 Suppl 1, pp. S88–94, 2013.

[38] SK Service, TM Teslovich, C Fuchsberger, V Ramensky, P Yajnik, DC Koboldt, DE Larson, Q Zhang, L Lin, R Welch, L Ding, MD McLellan, M O'Laughlin, C Fronick, LL Fulton, V Magrini, A Swift, P Elliott, M-R Jarvelin, M Kaakinen, MI McCarthy, L Peltonen, A Pouta, LL Bonnycastle, FS Collins, N Narisu, HM Stringham, J Tuomilehto, S Ripatti, RS Fulton, C Sabatti, RK Wilson, M Boehnke, and NB Freimer, "Re-sequencing expands our understanding of the phenotypic impact of variants at gwas loci," *PLOS GENETICS*, vol. 10, 2014.

[39] Li Shen and Yi Zhang, "5-hydroxymethylcytosine: generation, fate, and genomic distribution," *Current Opinion in Cell Biology*, vol. 25, no. 3, pp. 289 – 296, 2013.

[40] Hironori Waki, Toshimasa Yamauchi, and Takashi Kadowaki, "The epigenome and its role in diabetes," *Current Diabetes Reports*, vol. 12, no. 6, pp. 673–685, 2012.

[41] Jose Salavert Torres, Ignacio Blanquer Espert, Andres Tomas Dominguez, Vicente Hernendez, Ignacio Medina, Joaquin Terraga, and Joaquin Dopazo, "Using gpus for the exact alignment of short-read genetic sequences by means of the burrows-wheeler transform," *IEEE/ACM Trans. Comput. Biol. Bioinformatics*, vol. 9, no. 4, pp. 1245–1256, July 2012.

[42] Peter J A Cock, Christopher J Fields, Naohisa Goto, Michael L Heuer, and Peter M Rice, "The Sanger FASTQ file format for sequences with quality scores, and the Solexa/Illumina FASTQ variants," *Nucleic Acids Res*, vol. 38, no. 6, pp. 1767–1771, April 2010.

[43] Héctor Martínez, Joaquín Tárraga, Ignacio Medina, Sergio Barrachina, Maribel Castillo, Joaquín Dopazo, and Enrique S. Quintana-Ortí, "Concurrent and accurate rna sequencing on multicore platforms," Technical report icc 2013-03-01, Universitat Jaume I, Castellón, Spain, 2013.

[44] Joaquín Tárraga, Vicente Arnau, Héctor Martínez, Raúl Moreno, Diego Cazorla, José Salavert-Torres, Ignacio Blanquer-Espert, Joaquín Dopazo, and Ignacio Medina, "Acceleration of short and long dna read mapping without loss of accuracy using suffix array," *Bioinformatics*, Aug. 2014.

[45] M. Hassaballah, Saleh Omran, and Youssef B. Mahdy, "A review of simd multimedia extensions and their usage in scientific and engineering applications," *Computer J.*, vol. 51, no. 6, pp. 630–649, Nov. 2008.

[46] T. F. Smith and M. S. Waterman, "Identification of common molecular subsequences.," *Journal of molecular biology*, vol. 147, no. 1, pp. 195–197, Mar. 1981.

[47] H. Li and R. Durbin, "Fast and accurate short read alignment with burrows-wheeler transform.," *Bioinformatics*, vol. 25, no. 14, pp. 1754–60, 2009.

[48] Yuanxin Xi and Wei Li, "BSMAP: whole genome bisulfite sequence MAPping program.," *BMC bioinformatics*, vol. 10, no. 1, pp. 232+, 2009.

[49] Peter A. Jones, "Functions of dna methylation: islands, start sites, gene bodies and beyond," *Nature Reviews Genetics*, vol. 13, pp. 484–492, 2013.

[50] Nuno A. Fonseca, Johan Rung, Alvis Brazma, and John C. Marioni, "Tools for mapping high-throughput sequencing data," *Bioinformatics*, vol. 28, no. 24, pp. 3169–3177, 2012.

[51] Pao-Yang Chen, Shawn Cokus, and Matteo Pellegrini, "Bs seeker: precise mapping for bisulfite sequencing.," *BMC Bioinformatics*, vol. 11, pp. 203, 2010.

Sobre el paralelismo anidado de tareas en la factorización LU de Matrices Jerárquicas

Rocío Carratalá-Sáez[1] y Enrique S. Quintana-Ortí[2]

Resumen— **En este artículo se presenta una versión paralela de la factorización LU de Matrices Jerárquicas ($\mathcal{H}$-matrices) provenientes de Métodos de Elementos de Contorno (BEM). Estas matrices contienen estructuras internas cuya dimensión varía durante la ejecución de operaciones sobre las mismas, por lo que es necesario desligar las estructuras de datos de aquellas utilizadas para representar las dependencias en las tareas en las que se basa la implementación paralelizada. Utilizamos el modelo de programación OmpSs-2 y su runtime para determinar el flujo de datos intrínseco al paralelismo en tiempo de ejecución, así como para aprovechar las dependencias débiles de tareas y la "liberación temprana" (*early release*) de dependencias. Gracias a estas funcionalidades, puede acelerarse la ejecución de la versión paralela de la $\mathcal{H}$-LU y mejorarse el rendimiento.**

Palabras clave— **Matrices jerárquicas, factorización LU, paralelismo basado en tareas, procesadores multinúcleo, BEM.**

I. Introducción

LAS matrices jerárquicas (abreviadas como $\mathcal{H}$-matrices), combinadas con la $\mathcal{H}$-aritmética, ofrecen una abstracción matemática eficiente para lidiar con problemas provenientes de métodos de elementos de contorno, operadores elípticos parcialmente diferenciables o ecuaciones integrales [8]. Estas matrices permiten comprimir una matriz original de dimensión $n \times n$ utilizando solamente $O(n\ c\ log\ n)$ elementos, así como realizar operaciones de álgebra lineal, tales como la factorización LU, con un coste de $O(n\ c^2\ log^2\ n)$ operaciones en coma flotante (flops), siendo c es un parámetro que puede ajustarse para controlar la precisión de la aproximación [6], [7].

A lo largo de los últimos años, se han desarrollado bibliotecas para realizar operaciones de álgebra lineal con $\mathcal{H}$-matrices. Algunos de los trabajos más relevantes han dado lugar a los paquetes HLib[1], H2Lib[2] y HLibPro[3], que en su mayoría utilizan los núcleos computacionales definidos en *Basic Linear Algebra Subprograms* (BLAS) [4] para realizar los cálculos de las operaciones básicas del álgebra lineal. En principio, las rutinas de estos paquetes software pueden (o pueden ser fácilmente modificadas para) ejecutarse en paralelo en arquitecturas multicore enlazando una implementación de BLAS, tal como Intel MKL. No obstante, el grado de eficiencia paralela alcanzable de este modo está limitado, principalmente debido a que las $\mathcal{H}$-matrices están *dominadas* por bloques de rango bajo, lo cual hace necesario explotar el paralelismo en un nivel superior.

Aunque algunas de las bibliotecas actuales presentan algoritmos paralelos multihilo, estos se basan en OpenMP o Intel TBB (*thread building blocks*). El paralelismo de tareas se ha aplicado recientemente en la resolución de sistemas de ecuaciones lineales tanto densos [3] como dispersos [1], alcanzándose en ambos campos buenos resultados a nivel de rendimiento en arquitecturas multinúcleo. Dado que las $\mathcal{H}$-matrices residen en un escenario intermedio entre las matrices densas y las dispersas, parece natual aplicar técnicas similares en operaciones de $\mathcal{H}$-aritmética. En [2] presentamos un prototipo de la implementación de la factorización $\mathcal{H}$-LU para sistemas lineales densos utilizando OmpSs. En dicho trabajo se prueba la viabilidad de optar por una estrategia paralela basada en tareas, si bien hay limitaciones; por un lado, el algoritmo empleado es un prototipo que asume bloques densos o nulos (no se incluyó soporte para bloques de rango bajo); y, por otro, la versión del modelo de programación OmpSs utilizada es la anterior a OmpSs-2, la cual carecía de algunas funcionalidades que son clave en el presente artículo.

En este trabajo presentamos una versión paralela de la $\mathcal{H}$-LU utilizando el modelo de programación OmpSs-2 para lograr una implementación paralela de la $\mathcal{H}$-LU disponible en H2Lib, una biblioteca Open Source desarrollada en el Scientific Computing Group de la Universität zu Kiel.

La $\mathcal{H}$-LU requiere que el *runtime* detecte en tiempo de ejecución las tareas de manera dinámica y los núcleos computacionales pueden realizarse mediante una ejecución secuencial de BLAS (para el caso denso), pero hay algunos aspectos de esta operación que la diferencian de los trabajos sobre matrices densas y/o dispersas previamente referenciados. En primer lugar, la "naturaleza recursiva" de las $\mathcal{H}$-matrices, dificulta la correcta detección y planificación de las tareas. A esto se suma que los bloques de rango bajo pueden variar (disminuyendo o aumentando su tamaño) en tiempo de ejecución. Gracias a las nuevas funcionalidades incorporadas en el modelo de programación OmpSs-2, es posible lidiar con dichas particularidades y alcanzar una buena eficiencia paralela.

El resto del artículo se estructura como sigue. En la Sección II detallamos algunos fundamentos matemáticos de las $\mathcal{H}$-matrices y cómo se obtienen y representan estas en la biblioteca H2Lib. En la Sección III presentamos las nuevas funcionalidades ofrecidas por el modelo de programación OmpSs-2 y, en la Sección IV, describimos el algoritmo para la $\mathcal{H}$-LU y cómo aprovechar dichas funcionalidades en

[1]Dpto. de Ingeniería y Ciencia de los Computadores, Univ. Jaume I, e-mail: rcarrata@uji.es.

[2]Dpto. de Informática de Sistemas y Computadores, Univ. Politècnica de València, e-mail: quintana@disca.upv.es.

[1]http://www.hlib.org/

[2]http://www.h2lib.org/

[3]https://www.hlibpro.com/

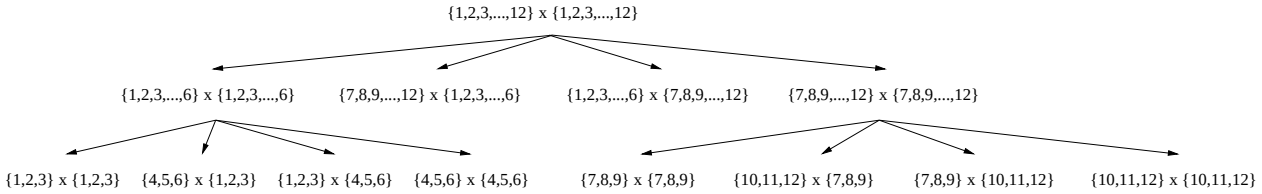


Fig. 1. *Block cluster tree* obtenido del conjunto de índices $I = \{1, 2, 3, ..., 12\}$.

una versión paralelizada del mismo. Finalmente, en la Sección V evaluamos el rendimiento de diferentes implementaciones paralelas y en la Sección VI resumimos las conclusiones obtenidas.

II. $\mathcal{H}$-matrices en H2Lib

A. Fundamentos de las $\mathcal{H}$-matrices

Para comprender cómo están representadas internamente en la biblioteca H2Lib las $\mathcal{H}$-matrices, presentamos brevemente algunos fundamentos matemáticos de las mismas. Para más detalles, véanse [7], [10].

El concepto abstracto de $\mathcal{H}$-matriz puede verse como una representación dispersa de los datos de una matriz densa que logra que los costes de almacenamiento se reduzcan en la medida en la que se explota la representación de los datos mediante bloques de rango bajo en forma factorizada, manteniendo el resto en bloques densos o de rango casi completo, dado que no hay beneficio al factorizarlos para su representación. Para obtener la jerarquía de bloques que define la estructura de la $\mathcal{H}$-matriz, debe utilizarse una condición de *admisibilidad*, la cual determina que un cierto bloque, que es admisible, puede aproximarse con una factorización de rango bajo, hasta una cierta precisión. Veamos esto.

Empecemos con la definición de *cluster tree* para un conjunto de índices dado.

Definición 1: Sea I un conjunto de índices con cardinalidad $n = 1$. El grafo $T_I = (E, V)$, donde V son los vértices y E las aristas, es un *cluster tree* sobre I, si I es la raíz de T_I y, $\forall\ v \in V$, o bien v es una hoja de T_I, o bien $v = \dot{\cup}_{v' \in S(v)} v'$, donde $S(v)$ representa el conjunto de descendientes directos de v.

Las $\mathcal{H}$-matrices representan un particionado jerárquico de un conjunto de índices en forma de *cluster tree*. Cuando se particiona el conjunto de índices producto $I \times I$, utilizando los subconjuntos de I definidos en T_I, obtenemos "*cluster trees* de *cluster trees*".

Definición 2: Sea T_I un *cluster tree* sobre I y considérese el nodo $b = p \times q$. El *block cluster tree* $T_{I\times I}$ sobre T_I puede definirse recursivamente para el nodo b, empezando por la raíz $I \times I$, como sigue:

$$S(b) = \begin{cases} \emptyset \text{ si } b \text{ es admisible o } S(p) = \emptyset \text{ o } S(q) = \emptyset, \\ S' \text{ en cualquier otro caso}, \end{cases}$$

donde $S' := \{p' \times q' : p' \in S(p), q' \in S(q)\}$.

La Figura 1 muestra un ejemplo de un *block cluster tree* cuyas hojas que conforman una partición $I \times I$, con $I = \{1, 2, 3, ..., 12\}$ utilizando un criterio de admisibilidad débil; véase [9] para más detalles.

El conjunto de matrices de rango bajo con un rango máximo k es un conjunto de $\mathcal{H}$-matrices.

Definición 3: El conjunto de $\mathcal{H}$-matrices para un *block cluster tree* $T_{I\times I}$ sobre un *cluster tree* T_I se define como:

$$\begin{aligned} \mathcal{H}(T_{I\times I}, k) := \{ & M \in \mathbb{R}^{I\times I} \mid \forall\ p \times q \in \mathcal{L}(T_{I\times I}) : \\ & rango(M\mid_{p\times q}) \leq k \vee \{p, q\} \\ & \cap\, \mathcal{L}(T_I) \neq \emptyset \} \end{aligned}$$

donde $\mathcal{L}(\mathcal{T}_\mathcal{I})$ es el conjunto de hojas de T_I y $k \in \mathbb{N}$.

Siguiendo con el ejemplo de la Figura 1, el *block cluster tree* definido en el ejemplo define el particionado de una matriz de tamaño 12×12 en una $\mathcal{H}$-matriz como la ilustrada en la Figura 2.

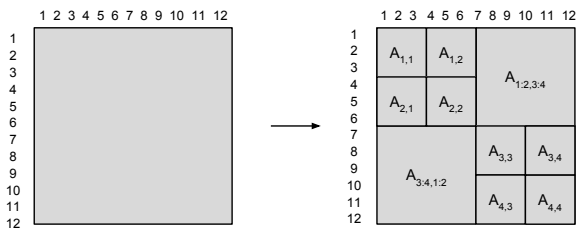


Fig. 2.$\mathcal{H}$-matriz obtenida aplicando el particionado definido por el *block cluster tree* de la Figura 1 sobre una matriz de tamaño 12×12.

Las $\mathcal{H}$-matrices utilizan la $\mathcal{H}$-aritmética para calcular la suma de matrices, su inversión, producto y la factorización LU con un coste computacional logarítmico [6]. Para esto, la suma de matrices de rango bajo se trunca para que tenga un rango c fijado (o, preferiblemente, con respecto a una precisión ϵ dada) mediante su descomposición en valores singulares (SVD) [5]. Dicho de otro modo, los resultados de las operaciones sobre $\mathcal{H}$-matrices (excepto el producto matriz-vector) son aproximados para poder garantizar una complejidad aritmética logarítmica.

B. Representación de $\mathcal{H}$-matrices en H2Lib

La biblioteca H2Lib ofrece rutinas secuenciales implementadas en C para el manejo de $\mathcal{H}$-matrices. Para poder determinar la partición que da lugar a una $\mathcal{H}$-matriz, es necesario clasificar en distintos conjuntos los grados de libertad (en adelante, DoFs) que definen el problema de origen, de manera que se pueda formar eficientemente un *cluster tree*. Asumiendo que conocemos la extensión geométrica de cada DoF que aparece en nuestra aplicación, esto se reduce frecuentemente al soporte de funciones básicas de elementos finitos. Gracias a esta información, podemos

configurar un cuadro delimitador de ejes paralelos $\mathcal{B}_t$ (o *bounding box*) que contenga la unión de todas las extensiones correspondientes al clúster t. Este cuadro, a su vez, se divide en dos partes a lo largo de alguna dimensión geométrica, lo que da lugar a dos cuadros separados $\mathcal{B}_{t_1}$, $\mathcal{B}_{t_2}$. A continuación, ambos cuadros se procesan de forma recursiva hasta que el número de DoFs ubicados en un cuadro es inferior a una constante prefijada, denotada por *tamaño de hoja* (C_{lf}) o *leafsize* (C_{lf}). Con el fin de manejar todos estos cuadros de manera eficiente, se organizan en una estructura de árbol (esto es, un *block cluster tree* anteriormente definido) expresada por `sonst` $= \{t_1, t_2\}$. Este proceso se resume en el Algoritmo 1.

Algoritmo 1 Construcción del *clustertree*

Entrada: Información geométrica sobre los DoFs almacenada en el vector `dofs`, de tamaño `size`.
Salida: Partición jerárquica de los DoFs vía el *clustertree* `t`.

```
procedure SETUP_CLUSTERTREE(dofs,size)
si size > C_lf entonces
  d ← FIND_SPLITTING_DIMENSION(dofs, size)
  sons ← 2
  t ← NEW_CLUSTER(dofs, size, sons)
  {dofs1, dofs2, size1, size2} ← SORT_DOFS(dofs, size, d)
  t_1 ← SETUP_CLUSTERTREE(dofs1, size1)
  t_2 ← SETUP_CLUSTERTREE(dofs2, size2)
  SONS(t) ← {t_1, t_2}
si no
  t ← NEW_LEAF_CLUSTER(dofs, size)
fin si
devolver t
```

En H2Lib, los distintos clústeres que conforman las particiones jerárquicas se representan con la estructura de datos (en C) que sigue:

```
struct cluster {
    uint      size;
    uint      *dofs
    uint      *bbox_min;
    uint      *bbox_max;
    cluster   *son;
    uint      sons;
}
```

Aquí, `size` es el total de alementos asociados al clúster y `dofs` es el vector que contiene los DoFs. El cuadro delimitador $\mathcal{B}_t$ para un clúster t se almacena entre los vectores `bbox_min` y `bbox_max`, respectivamente. Por su parte, `son` y `sons` representan el árbol de clústers.

La condición de admisibilidad empleada en H2Lib es:

$$\max\{\mathrm{diam}(\mathcal{B}_t), \mathrm{diam}(\mathcal{B}_s)\} \leq \eta \, \mathrm{dist}(\mathcal{B}_t, \mathcal{B}_s),$$

donde *diam* y *dist* hacen referencia al diámetro Euclideano del cuadro delimitador y la distancia Euclideana entre dos de ellos y $\eta \in \mathbb{R}_{>0}$ es el parámetro elegido para controlar el canje entre la cantidad de bloques admisibles en la matriz y la exactitud de la aproximación.

Finalmente, el particionado de la $\mathcal{H}$-matriz se hace de forma recursiva aplicando el Algoritmo 2 (una explicación detallada del mismo puede encontrarse en [7], [6]). En él, en un determinado nivel de la recursión, pueden producirse tres tipos de estructuras: un bloque de rango bajo (i.e., un nuevo bloque admisible), una nueva partición recursiva (vía una llamada al mismo algoritmo) o un bloque denso convencional (i.e., un bloque no admisible).

Algoritmo 2 Construcción del *block cluster tree*

Entrada: Clúster fila `t`, clúster columna `s`.
Salida: *blocktree* `b` para el par de clústers `(t,s)`.

```
procedure SETUP_BLOCKTREE(t,s)
si ADMISSIBLE(t, s) entonces
  b ← NEW_ADMISSIBLE_BLOCK(t, s)
si no
  si SONS(t) ≠ ∅ ∧ SONS(s) ≠ ∅ entonces
    b ← NEW_PARTITIONED_BLOCK (t, s)
    para todo t' ∈ SONS(t), s' ∈ SONS(s) hacer
      b[t'][s'] ← SETUP_BLOCKTREE(t', s')
    fin para
    devolver b
  si no
    b ← NEW_INADMISSIBLE_BLOCK(t, s)
  fin si
fin si
devolver b
```

La estructura con la que se representa una $\mathcal{H}$-matriz proveniente del proceso recursivo descrito en H2Lib (en C) tiene esta forma:

```
struct hmatrix {
    cluster  rc, cc;
    rkmatrix r;
    amatrix  f;
    hmatrix  *son;
    uint     rsons, csons;
}
```

En esta estructura, `rc` y `cc` son, respectivamente, los clústers de filas y columnas del bloque en concreto; `rsons` y `csons` representan la cantidad de hijos por fila y columna que respectivamente presenta dicho bloque (nótese que ambos serán 0 si el bloque es una hoja en el *block cluster tree*). Los bloques de rango bajo se almacenan en una `rkmatrix`, mientras que los densos en una `amatrix` y los bloques sobre los que se aplica de nuevo un particionado de manera recursiva se representan con un vector de punteros que referencian a los bloques hijos.

Por su parte, la estructura diseñada en H2Lib (en C) para almacenar un bloque de rango bajo es:

```
typedef struct rkmatrix {
        uint k;         /* Maximal rank */
        amatrix A;      /* Left factor A */
        amatrix B;      /* Right factor B */
}
```

III. El modelo OmpSs-2

El modelo de programación OpenMP 4.5, con el que OmpSs presenta ciertas similitudes, ofrece soporte para anidación y también para definir depen-

dencias entre tareas. No obstante, cuando se definen dependencias anidadas, es necesario aplicar medidas correctoras para asegurar la correcta coordinación de las dependencias entre los distintos niveles. Por ejemplo, es necesario añadir un punto de sincronización (mediante `taskwait`) al final de cada una de las tareas que está particionada en subtareas. Esto limita la eficiencia máxima que se puede lograr.

Por su parte, OmpSs-2 presenta dos características nuevas, que no están disponibles en OpenMP, muy útiles para abordar el problema en el que se basa el trabajo que se describe en este artículo: dependencias *débiles* y "liberación temprana" (*early release*) de dependencias.

Las dependencias respecto a operandos de una determinada tarea anotadas como *débiles* son ignoradas por el *runtime* cuando se determina si una tarea está lista para ejecutarse o no. Esto es posible porque dichos operandos solo son leídos o escritos por tareas secundarias y no por la tarea en la que se anotan dependencias respecto a ellos de forma débil. La ventaja de utilizar este tipo de dependencias es que las subtareas se pueden instanciar antes y en paralelo.

La liberación temprana de dependencias permite proporcionar dependencias de grano más fino entre tareas "hermanas". Esto se consigue liberando inmediatamente las dependencias de una tarea, en cuanto finaliza, siempre que no estén siendo utilizadas en ese momento por ninguna de sus subtareas. Además, tan pronto como las subtareas terminan, se liberan las dependencias que no están siendo utilizadas en ese momento por ninguna de sus tareas "hermanas".

En la Sección IV se detalla cómo afectan estas dos funcionalidades al paralelismo de tareas aplicado en la $\mathcal{H}$-LU.

IV. Factorización LU de $\mathcal{H}$-matrices

A. Algoritmo secuencial de la $\mathcal{H}$-LU

El algoritmo para la factorización LU de $\mathcal{H}$-matrices puede entenderse como una generalización del algoritmo a *Blocked Right Looking LU* que, en este caso, tiene en cuenta una estructura jerárquica [5]. Consideremos la $\mathcal{H}$-matriz $A \in \mathbb{R}^{n\times n}$ particionada del mismo modo que en la Figura 1; la secuencia de operaciones que calcula la $\mathcal{H}$-LU en ese caso es:

O1.1 :	$A_{1,1}$	:=	$L_{1,1}U_{1,1}$,
O1.2 :	$U_{1,2}$	:=	$L_{1,1}^{-1}A_{1,2}$,
O1.3 :	$L_{2,1}$	:=	$A_{2,1}U_{1,1}^{-1}$,
O1.4 :	$A_{2,2}$	:=	$A_{2,2} - L_{2,1}, \cdot U_{1,2}$,
O1.5 :	$A_{2,2}$	:=	$L_{2,2}U_{2,2}$,
O2 :	$U_{1:2,3:4}$	:=	$L_{1:2,1:2}^{-1}A_{1:2,3:4}$,
O3 :	$L_{3:4,1:2}$	:=	$A_{3:4,1:2}U_{1:2,1:2}^{-1}$,
O4 :	$A_{3:4,3:4}$	:=	$A_{3:4,3:4} - L_{3:4,1:2} \cdot U_{1:2,3:4}$,
O5.1 :	$A_{3,3}$	:=	$L_{3,3}U_{3,3}$,
O5.2 :	$U_{3,4}$	:=	$L_{3,3}^{-1}A_{3,4}$,
O5.3 :	$L_{4,3}$	:=	$A_{4,3}U_{3,3}^{-1}$,
O5.4 :	$A_{4,4}$	:=	$A_{4,4} - L_{4,3}, \cdot U_{3,4}$,
O5.5 :	$A_{4,4}$	:=	$L_{4,4}U_{4,4}$.

En concreto, se resuelven las siguientes operaciones básicas cuando se opera con bloques densos:

- Factorización LU (en el caso del ejemplo, en O1.1, O1.5, O5.1 y O5.5);
- Resolución de un sistema de ecuaciones lineal triangular (en el caso del ejemplo, con un factor triangular inferior unitario en O1.2, O2 y O5.2, o con un factor triangular superior en O1.3, O3 y O5.3);
- Producto matriz-matriz (en el caso del ejemplo, en O1.4, O4 y O5.4).

En el caso de haber bloques de rango bajo involucrados, que estarán representados en forma factorizada, esto es, $X = AB^*$ donde A, B tienen solamente k columnas (rango de X), las operaciones básicas para su manejo son:

- Producto matriz-vector;
- Resolución de un sistema de ecuaciones lineal triangular aplicando sustitución progresiva o regresiva a las k columnas de A o k filas de B, respectivamente;
- Producto matriz-matriz.

Adicionamente debe tenerse en cuenta que, cuando se suman dos bloques de rango bajo, se vuelve a comprimir el resultado utilizando la descomposición en valores singulares para descartar los valores más pequeños.

B. Dependencias entre las operaciones de la $\mathcal{H}$-LU

La Figura 3 representa las dependencias existentes entre las operaciones que componen la factorización $\mathcal{H}$-LU descrita en la Sección III para la matriz de ejemplo A. En concreto, puede observarse que la factorización se descompone inicialmente en 5 tareas (O1 - O5), y algunas de ellas, a su vez, se descomponen en subtareas (por ejemplo, la tarea O1 se descompone en 5 tareas: O1.1 - O1.5).

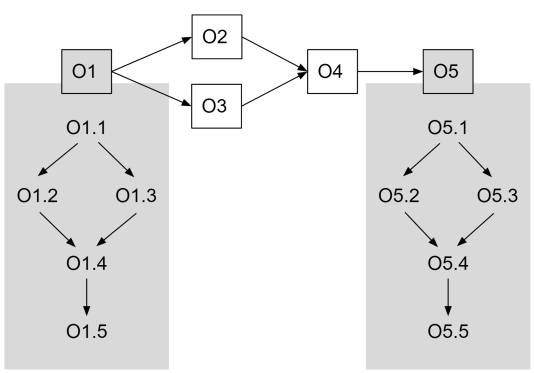


Fig. 3.Dependencias de datos entre las operaciones de la factorización $\mathcal{H}$-LU para el algoritmo descrito en la Sección III para la matriz de ejemplo A.

Aparentemente, el grado de paralelismo está limitado, dado que solamente podrían ejecutarse en paralelo O1.2 con O1.3, O2 con O3 y O5.2 con O5.3. No obstante, esto se debe al exceso de simplicidad de la matriz tomada como ejemplo. Las $\mathcal{H}$-matrices presentan particionados más complejos (es-

to es, anidándose varios particionados) y que no solamente afectan a los bloques diagonales. La Figura 4 presenta un ejemplo (todavía simple) de una $\mathcal{H}$-matriz alternativa a la presentada anteriormente.

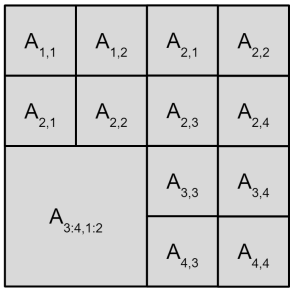


Fig. 4.$\mathcal{H}$-matriz alternativa con un particionado simple de $2x2$, dimensión 12.

En este nuevo ejemplo, la secuencia de operaciones que componen el algoritmo para la $\mathcal{H}$-LU descrito anteriormente es ahora más compleja. Concretamente, la operación

$$\text{O2}: \quad U_{1:2,3:4} \quad := \quad L^{-1}_{1:2,1:2} A_{1:2,3:4}$$

puede ahora subdividirse en las siguientes operaciones:

$$\begin{aligned}
\text{O2.1}: \quad U_{1,3} &:= L^{-1}_{1,1} A_{1,3}, \\
\text{O2.2}: \quad U_{1,4} &:= L^{-1}_{1,1} A_{1,4}, \\
\text{O2.3}: \quad A_{2,3} &:= A_{2,3} - L_{2,1} \cdot U_{1,3}, \\
\text{O2.4}: \quad A_{2,4} &:= A_{2,4} - L_{2,1} \cdot U_{1,4}, \\
\text{O2.5}: \quad U_{2,3} &:= L^{-1}_{2,2} A_{2,3}, \quad \text{y} \\
\text{O2.6}: \quad U_{2,4} &:= L^{-1}_{2,2} A_{2,4}.
\end{aligned}$$

Si bien O1 y O2 siguen presentando la dependencia de datos que ya se indicó en la Figura 3, sus respectivas subtareas tienen, entre ellas, parte de dichas dependencias (aunque no su totalidad), tal como se representa en la Figura 5.

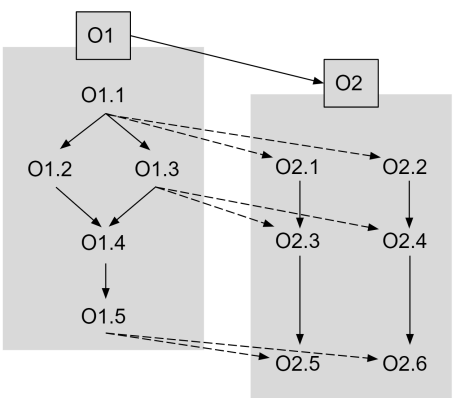


Fig. 5.Dependencias de datos entre las operaciones O1 y O2 de la factorización $\mathcal{H}$-LU para el algoritmo descrito en la Sección III para la nueva matriz de ejemplo (Figura 4). Con líneas discontinuas se indican las dependencias entre tareas "no hermanas" (no son subtareas de una misma tarea con granularidad inmediatamente superior).

Así pues, si en este escenario se aplica paralelismo anidado, tras la operación O1.1 pueden no solo realizarse O1.2 y O1.3, sino también O2.1 y O2.2.

C. *OmpSs-2 aplicado a la $\mathcal{H}$-LU*

Con los modelos de programación OmpSs (en su primera versión) y OpenMP 4.5, es necesario incluir un `taskwait` al final de cada tarea que se encuentra subdividida en subtareas para garantizar una correcta ejecución si se aplica paralelismo anidado. La ventaja de aplicar, en su lugar, el modelo de programación OmpSs-2 es que pueden anidarse las dependencias de tareas hasta alcanzar aquellas de grano más fino, anotándose como dependencias *débiles* todas aquellas relativas a tareas de grano grueso (en el ejemplo, las presentes entre O1 y O2, entre otras) y solamente como dependencias *fuertes* las existentes entre las tareas de grano más fino (en el ejemplo, O1.1 y O2.1, entre otras). De este modo, pueden solaparse ejecuciones pertenecientes a distintos niveles de la anidación de tareas y anticiparse parte de las ejecuciones en cuanto las dependencias estrictamente necesarias se satisfacen (y no todas las indicadas en las tareas de grano más grueso de las que son subtareas las que ya están listas para ejecutarse) gracias a la liberación temprana de dependencias.

V. Experimentos

En esta sección se describen los experimentos realizados para comprobar la eficiencia de la versión paralelizada de la $\mathcal{H}$-LU presente en H2Lib explotando paralelismo de tareas. Las $\mathcal{H}$-matrices utilizadas en los test pertenecen al contexto de BEM y, en particular, de la resolución de la ecuación de Laplace en $d \in \{2, 3\}$ dimensiones, con los núcleos computacionales de la forma

$$g : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}\,,$$

$$g(x,y) = \begin{cases} -\frac{1}{2\pi} \log \|x-y\|_2 & : d = 2, \\ \frac{1}{4\pi} \|x-y\|_2^{-1} & : d = 3. \end{cases}$$

Se ha utilizado aritmética de doble precisión IEEE 754 en un nodo del supercomputador MareNostrum 4, en el Barcelona Supercomputing Center, que contiene dos sockets Intel Xeon Platinum 8160, con 24 núcleos por socket y 96 Gbytes de memoria DDR4 RAM. Además, se han empleado `gcc` 4.8.5, Intel MKL 2017.4 (con las instrucciones AVX2 habilitadas), y OmpSs-2 (`mcxx` 2.1.0).

El criterio de admisibilidad utilizado es:

$$\max\{\operatorname{diam}(\mathcal{B}_t), \operatorname{diam}(\mathcal{B}_s)\} \leq \eta \operatorname{dist}(\mathcal{B}_t, \mathcal{B}_s)\,,$$

donde $\eta \in \mathbb{R}_{>0}$.

Nuestros experimentos demuestran los beneficios de utilizar las nuevas funcionalidades de OmpSs-2 (dependencias *débiles* y liberación temprana de dependencias) en el paralelismo de tareas anidado aplicado a la resolución de la $\mathcal{H}$-LU. Para esto, utilizaremos dos configuraciones de $\mathcal{H}$-matrices: una de dimensión 30K proveniente de un problema de BEM

con $d = 2$, y otra de dimensión 42K donde el problema de origen tiene $d = 3$. En ambas, el grado de *dispersión* está controlado con el parámetro η, que para el primer caso tomará los valores 0,25, 0,5 y 1, y, para el segundo, 0,5, 1, 2. La Figura 6 presenta las seis $\mathcal{H}$-matrices utilizadas.

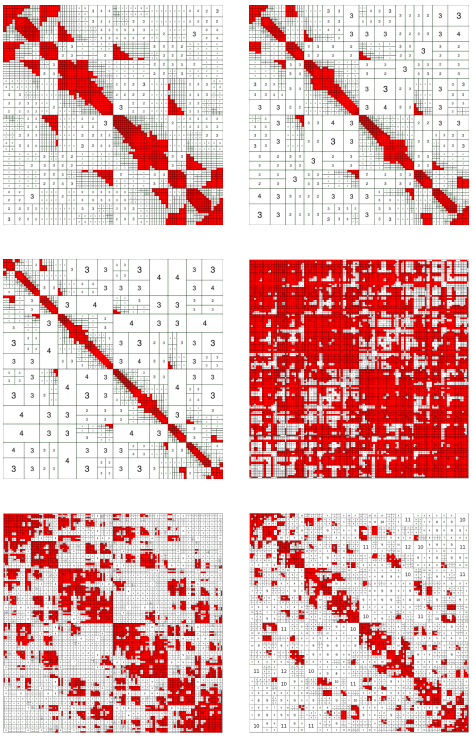


Fig. 6.$\mathcal{H}$-matrices utilizadas para los experimentos. De arriba a abajo y de izquierda a derecha, las tres primeras tienen dimensión 30K y los valores de η son 0,25, 0,5 y 1; las tres restantes tienen dimensión 42K y sus valores de η son 0,5, 1 y 2. Los bloques rellenados con color rojo son densos, el resto son bloques de rango bajo.

En los experimentos realizados con las matrices de dimensión 30K se utilizarán hasta 24 cores y, con las de dimensión 42K, hasta 48 cores. En todos los casos, se comparan los rendimientos paralelos utilizando dependencias *débiles* (WD) y sin utilizarlas. Las Figuras 7 y 8 presentan los *speedup* alcanzados en los experimentos realizados, en relación a la ejecución secuencial, representándose con líneas discontinuas los valroes correspondientes a las implementaciones en las que no se incluyen WD.

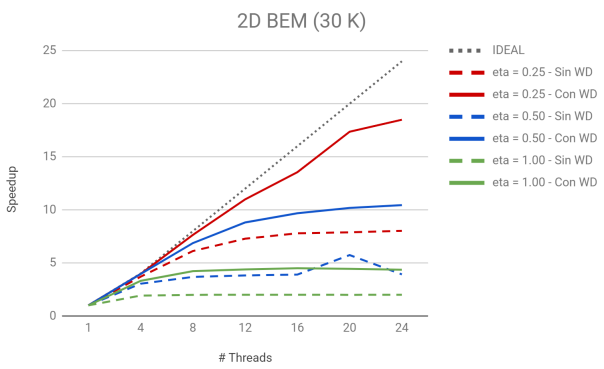


Fig. 7.Speedup de la ejecución paralela de la $\mathcal{H}$-LU con y sin dependencias débiles (WD), para las matrices de dimensión 30K con distintos valores de η y hasta 24 cores.

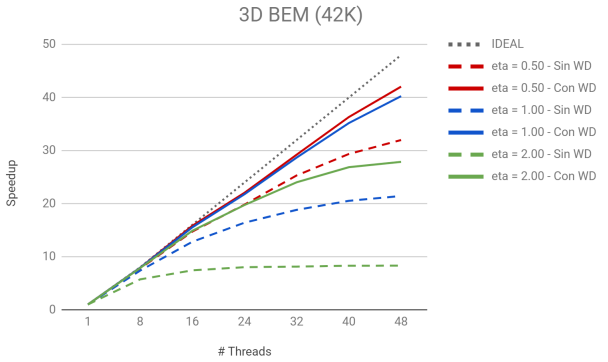


Fig. 8.Speedup de la ejecución paralela de la $\mathcal{H}$-LU con y sin dependencias débiles (WD), para las matrices de dimensión 42K con distintos valores de η y hasta 48 cores.

En general, se observa que el *speedup* aumenta con el ratio de bloques densos, reportándose valores notablemente altos para $d = 3$ (dado que la cantidad de núcleos no es demasiado alta comparada con la dimensión del problema) y menores para el caso $d = 2$. Estas diferencias en términos de eficiencia pueden justificarse analizándose la estructura de las $\mathcal{H}$-matrices. Concretamente, en el caso de $d = 3$, hay una mayor cantidad de bloques, lo cual aumenta el número de tareas a ejecutar y ayuda a exponer un mayor grado de concurrencia en ese caso.

VI. Conclusiones

Hemos mostrado eficiencias paralelas destacables para la factorización $\mathcal{H}$-LU de $\mathcal{H}$-matrices provenientes de BEM en 2D y 3D. Esto ha sido posible, en gran parte, gracias a utilizar las nuevas características presentes en OmpSs-2 (dependencias *débiles* y liberación temporana de dependencias), las cuales permiten explotar el paralelismo de tareas anidado. Como parte del trabajo futuro, queremos investigar esquemas de paralelismo híbridos que combinen paralelismo de tareas con paralelismo multihilo a nivel de núcleos computacionales, así como nuevas estrategias para extraer un mayor grado de paralelismo de tareas.

Agradecimientos

Los investigadores de la Universitat Jaume I (UJI) han sido financiados con los proyectos CICYT TIN2014-53495-R y TIN2017-82972-R del MINECO y FEDER; el proyecto UJI-B2017-46 de la UJI; y el programa FPU del MECD.

Referencias

[1] José I. Aliaga, Rosa M. Badia, María Barreda, Matthias Bollhöfer, and Enrique S. Quintana-Ortí, *Leveraging task-parallelism with OmpSs in ILUPACK's preconditioned cg method*, 26th Int. Symp. on Computer Architecture and High Performance Computing (SBAC-PAD 2014), pages 262–269, 2014.

[2] José I. Aliaga, Rocío Carratalá-Sáez, Ronald Kriemann, Enrique S. Quintana-Ortí *Task-parallel LU factorization of hierarchical matrices using OmpSs*, 2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), 2017, pp. 1148–1157. doi:10.1109/IPDPSW.2017.124.

[3] Rosa M. Badia, Jose R. Herrero, Jesus Labarta, Jose M. Pérez, Enrique S. Quintana-Ortí, and Gregorio Quintana-Ortí. *Parallelizing dense and banded linear algebra libra-*

ries using SMPSs, Concurrency and Computation: Practice and Experience, 21:2438–2456, 2009.

[4] Jack J. Dongarra, Jeremy Du Croz, Sven Hammarling, and Iain Duff. *A set of level 3 basic linear algebra subprograms*, ACM Trans. on Mathematical Software, 16(1):1–17, March 1990.

[5] Gene H. Golub and Charles F. Van Loan. *Matrix Computations*, The Johns Hopkins University Press, Baltimore, 3rd edition, 1996

[6] Lars Grasedyck and Wolfgang Hackbusch. *Construction and arithmetics of $\mathcal{H}$-matrices*, Computing, 70(4):295–334, August 2003.

[7] Wolfgang Hackbusch. *A sparse matrix arithmetic based on H -matrices. part i: Introduction to H -matrices*, Computing, 62(2):89–108, May 1999.

[8] Wolfgang Hackbusch. *Hierarchical Matrices: Algorithms and Analysis*, volume 49 of Springer Series in Computational Mathematics. Springer-Verlag Berlin Heidelberg, 2015.

[9] Mohammad Izadi. *Hierarchical matrix techniques on massively parallel computers*, Ph.D. dissertation, Universität Leipzig, 2012.

[10] Ronald Kriemann. *H -LU factorization on many-core systems*, Computing and Visualization in Science, 16(3):105– 117, 2013.

[11] OmpSs project home page. *http://pm.bsc.es/ompss.*

[12] The OpenMP API specification for parallel programming. *http://www.openmp.org/.*

Copositividad de una matriz: retos computacionales

E.M.T. Hendrix[1], L.G. Casado[2] y B. G.-Tóth[3]

Resumen— **Encontrar el mínimo de un problema de optimización cuadrática estándar permite determinar que una matriz simétrica es copositiva. El espacio de búsqueda del valor mínimo se realiza en un simplex unidad. En las jornadas SARTECO'18 se presentó un algoritmo que evalúa las caras de un simplex unidad para determinar si una matriz simétrica es copositiva. El problema es computacionalmente costoso debido que el número de caras es 2^n-1 en un simplex unidad de dimensión n. Este algoritmo hace una búsqueda de arriba hacia abajo en el grafo de caras (de mayor a menor dimensión). Otros estudios se basan en encontrar una cara con mayor dimension donde la función es convexa en cada uno de sus lados (caras de dimension 2). Esta ultima aproximación se muestra muy eficiente cuando la cara convexa de mayor dimensión tiene una dimensión media-baja en relación a n. En este trabajo se estudian los retos computacionales de estos algoritmos para su posible paralelización.**

Palabras clave— **Optimización cuadrática estándar, Matriz copositiva, simplex unidad, cara.**

I. Introducción

EN este trabajo se estudian dos problemas directamente relacionados: el de programación cuadrática estándar (StQP) y el de determinar si una matriz simétrica es copositiva. Se define el simplex unidad de dimension n como

$$\Delta = \{x \in \mathbb{R}^n_+ \mid \sum_{i=1}^{n} x_i = 1\}. \quad (1)$$

Dada una matriz simétrica A de $n \times n$, se define el problema StQP como

$$f^* := \min_{x \in \Delta} f(x) := x^T Ax. \quad (2)$$

Aunque en [1] se define el problema StQP como un problema de maximización, se puede convertir en uno de minimización teniendo en cuenta $-f$.

El StQP aparece en muchas aplicaciones, como en la optimización de portafolio [2], la genética de poblaciones [3], problemas de asignación de recursos [4] y la teoría evolutiva del juego, con interés en la teoría económica [5], entre otras. Existen estudios que proponen algoritmos específicos para resolver StQP [6], [7], [8], [9]. Estos trabajos muestran que la eficiencia de los algoritmos propuestos se reduce conforme se incrementa la densidad (dimensión) de las caras convexas del simplex unidad. Estas caras convexas están determinadas por la matriz A y son las que pueden contener los minimizadores globales que hay que encontrar para resolver (2).

[1]Dpto. de Arquitectura de Computadores, Universidad de Málaga, e-mail: eligius@uma.es.
[2]Dpto. de Informática, Universidad de Almería, ceiA3, e-mail: leo@ual.es
[3]Dpt. Computational Optimization, University of Szeged, Hungary, e-mail: boglarka@inf.szte.hu

Una matriz simétrica A de $n \times n$ se certifica que es copositiva cuando

$$\forall x \in \Delta,\ x^T Ax \geq 0, \quad (3)$$

(ver eq. (1)). Se puede determinar si una matriz es semidefinida positiva (PSD) mediante sus autovalores, pero no su copositividad, que es un problema NP-Completo [10]. Resolver StQP también resuelve el problema de determinar la copositividad de la matriz asociada, ya que si el valor mínimo $f^* < 0$ en (2) entonces A no es copositiva y si $f^* \geq 0$ entonces A es copositiva.

Este trabajo se organiza de la siguiente forma. En la sección II se describe el enfoque basado en las caras para demonstrar la copositividad de la matriz A. La sección III describe los algoritmos desarrollados. La sección IV muestra los resultados computacionales preliminares para motivar la necesidad de la computación paralela. Finalmente, las conclusiones se muestran en la sección V.

II. B&B espacial (BBE) versus algoritmos basados en caras.

El algoritmo de Ramificación y Acotación (B&B, por sus siglas en Inglés) presentado en [11] permite determinar si una matriz no es copositiva o que es ϵ-copositiva. En el Lema 2 de [11] se establece que dado $\Delta_k \subseteq \Delta$ con $\Delta_k = conv\{v_1, \ldots, v_n\}$, si

$$v_i^T Av_j \geq 0,\ \forall i, j = 1, \ldots, n \quad (4)$$

entonces $x^T Ax \geq 0$, $\forall x \in \Delta_k$. Además, se establece el punto de división de un lado de Δ_k de forma que los nuevos lados generados en la división verifiquen $v_i^T Av_j \geq 0$, estableciendo así el refinamiento del subsimplex.

Mediante el refinamiento de símplices, la certificación de la ϵ-copositividad requiere mucha más computación que verificar que una matriz no es copositiva. En [12], donde se usó computación grid para ejecutar una versión mejorada del algoritmo BBE mostrado en [11], se pudo determinar que matrices con varios miles de dimensiones no eran ϵ-copositivas. Sin embargo, solo se pudo determinar la ϵ-copositivad de matrices de dimensiones menores o iguales a $n = 22$ en un tiempo razonable (5 horas). Esto hace que este tipo de problemas sean interesantes para la computación paralela ya que los subsimplices pueden evaluarse de forma independiente.

El inconveniente de los algoritmos BBE es que se evalúan puntos que no necesariamente coinciden con mínimos locales de (2) y no se usa información de las derivadas para explotar la monotonicidad [13].

Existen varios estudios que usan explícitamente las condiciones de primer orden de los mínimos locales de (2) [14], [15], [8], [9]. Para un óptimo local $x \geq 0$ de (2) existen valores de las variables duales μ y $\lambda_i \geq 0$ que satisfacen

$$Ax = \mu\mathbf{1} + E\lambda,\ x^T\lambda = 0, \tag{5}$$

donde $\mathbf{1}$ es el vector de todos los elementos uno, E la matriz identidad y $x_i\lambda_i = 0$ se denomina complementaridad. Se puede solucionar el problema (2) con cientos de variables usando Mixed Integer Programming (MIP), mediante una transformación del problema usando variable binarias [8].

Por otro lado, [9] muestra un algoritmo de BBE que enumera implícitemente todas las soluciones de las condiciones de primer orden usando cotas obtenidas mediante expresiones lineales y convexas.

A diferencia de los algoritmos BBE, donde un subsimplex Δ_k puede tener vértices con valores diferentes de cero o uno, para los algoritmos basados en caras, los sub-símplices son caras. Una cara $\Delta_k \subseteq \Delta$ se caracteriza por el indice k, donde la cadena de bits de la reprensentación binaria de k, $k_{(2}$, indica los elementos activos, es decir, aquellas componentes que pueden tomar valores distintos de cero ($x_i > 0$). La figura 1 muestra el grafo con el conjunto de caras para n=5. El nivel o número de elementos ℓ de una cara esta determinado por $\ell = \sum_{i=1}^{n}(k_{(2)})_i$. Existen $\binom{n}{\ell}$ caras en cada nivel ℓ. En la figura 1 se muestra la relación de una cara con las caras del nivel superior (se añade una componente activa) y las del nivel inferior (se quita una componente activa). Un *faceta* Δ_m de una cara Δ_k en nivel ℓ es una cara $\Delta_m \subset \Delta_k$ en nivel $\ell - 1$. El número de caras en el grafo es 2^n-1=$\sum_{\ell=1}^{n}\binom{n}{\ell}$. Para evaluar f en la cara Δ_k se considera la submatriz A_k de A.

Definición 1: Dada una matriz simétrica A de $n \times n$ y $k \in \mathbb{N}^+$, A_k es una matriz de $\ell \times \ell$ con $\ell \leq n$ y las filas y columnas i de A que cumplen $(k_{)2})i = 1$.

Centrándonos en las condiciones de segundo orden, [14], [15] y [9] usan la existencia de un mínimo en el interior relativo de una cara Δ_k donde f es convexa. La función cuadrática f es convexa en Δ_k, cuando la matriz A_k es semidefinida positiva (PSD). Un mínimo en el interior relativo $rint(\Delta_k)$ de una cara Δ_k es único cuando f es estrictamente convexa en Δ_k. Para que se cumpla lo anterior, es suficiente que A_k sea una matriz definida positiva, pero en [13] se demuestra que no es una condición necesaria. Por ejemplo, la matriz

$$A = \begin{pmatrix} 0 & -1 & -2 \\ -1 & 1 & -3 \\ -2 & -3 & 2 \end{pmatrix} \tag{6}$$

define una función f que es estrictamente convexa en Δ, pero no es definida positiva. Sea $D_\ell = E_\ell - \frac{1}{n}\mathbf{1}\mathbf{1}^T$. El mínimo interior x^* en Δ_k está determinado por

$$A_k x^* - \mu\mathbf{1} = 0,\ \mathbf{1}^T x^* = 1. \tag{7}$$

lo que implica que $D_\ell A_k x^* = 0$ [13]. Dada la dirección $r = D_\ell y$, se cumple que $\mathbf{1}^T(x^* + r) = 1$. Para el punto $x^* + r$ en una cara

$$f(x^* + r) = (x^* + r)^T A_k (x^* + r) = \tag{8}$$

$$f(x^*) + r^T A_k r + 2r^T D_\ell A_k x^*. \tag{9}$$

Como $D_\ell A_k x^* = 0$,

$$f(x^* + r) = f(x^*) + y^T D_\ell A_k D_\ell y. \tag{10}$$

Entonces, x^* es un punto mínimo cuando la matriz

$$H_k = D_\ell A_k D_\ell \tag{11}$$

es PSD. Concretamente, H_k debe ser definida positiva en las direcciones r con $\mathbf{1}^T r = 0$. Es decir, H_k tiene un solo autovalor nulo, con un autovector igual a $\mathbf{1}$, siendo positivos los demás autovalores.

En [14] se demuestra que f es estrictamente convexa en lado (i, j) de Δ cuando

$$A_{ii} + A_{jj} - 2A_{ij} > 0. \tag{12}$$

Definición 2: Se define el *grafo de lados convexos* $G = (N, C)$ de una matriz A con $N = \{1, \ldots, n\}$ y el lado $(i, j) \in C$ cuando se cumple (12), i.e. f es estrictemente convexo en el lado entre e_i y e_j [14]. La dimensión de G podría reducirse si existe un vértice que no cumple (12) con los demás vértices, pero habría que tener en cuenta la correspondencia entre los elementos de G y A. El grafo G_k se define de forma similar a A_k (ver def. 1). Por lo tanto, una condición necesaria pero no suficiente para que f sea convexa en la cara Δ_k es que el grafo asociado a A_k de lados convexos G_k sea completo. Para la matriz en (13), se cumple la condición necesaria, es decir, A_k genera un grafo G_k completo, pero esta condición no es suficiente para que f sea convexa en Δ_k, ya que ni A_k ni H_k son PSD.

$$A_k = \begin{pmatrix} 8 & 5 & 3 \\ 5 & 3 & 3 \\ 3 & 3 & 4 \end{pmatrix} \tag{13}$$

Considerando si $\Delta_k \subset \Delta_m \rightarrow f_k^* \geq f_m^*$, el algoritmo propuesto en [14] esta basado en encontrar las caras Δ_k con grafos G_k completos en el mayor nivel ℓ posible. Este es el problema de enumerar los cliques máximos en G. En [14], el algoritmo realiza una búsqueda de abajo hacia arriba (de menor a mayor nivel) en el grafo de caras (ver Fig. 1), donde se calculan límites inferiores y superiores de la caras para determinar si se puede mejorar el valor del mínimo actual. Para cada cara, antes de evaluarla, se verifica que una cara es un clique, i.e. su G_k asociado es completo.

III. Test de copositividad basado en caras

En [15] se muestran tres variantes de un algoritmo que realiza una búsqueda sistemática de un punto negativo de f en las caras de Δ para determinar si A es o no ϵ-copositiva. Dada una matriz simétrica A, primero se puede determinar que A es copositiva si cumple que

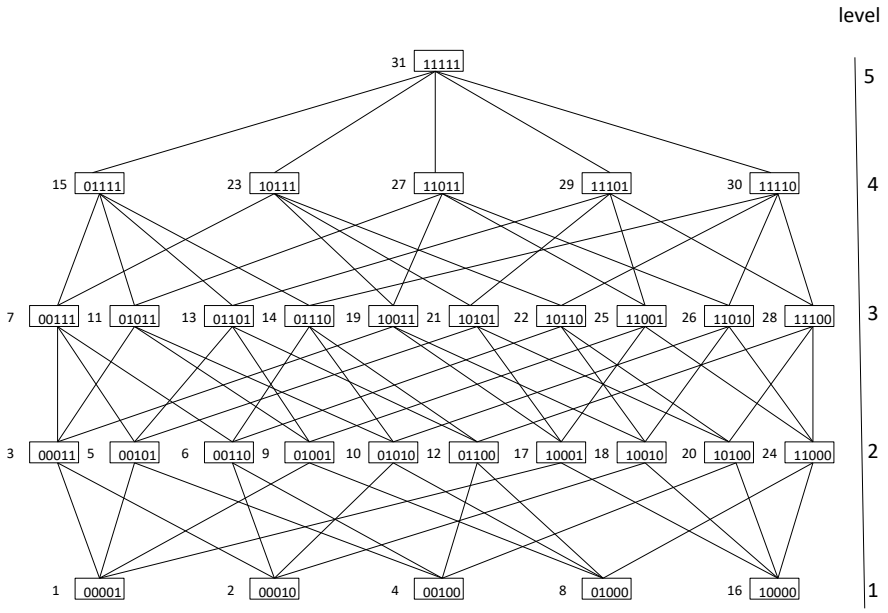


Fig. 1
Grafo de caras para $n = 5$.

- $\forall i, j,\ A_{ij} \geq 0$,
- A es PSD (descomposición de Cholesky).

Después se realizan los siguientes pasos:

- Se verifica que $A_{ii} \geq 0$, i.e f es positiva en caras que se corresponden con vértices de Δ ($\ell = 1$).
- Se verifica que $f(c) \geq 0$, $c = \frac{1}{n}\mathbf{1}$, i.e f es positiva en el centroide de Δ.
- Se construye G (ver la definición 2) que identifica los lados de Δ en los que f es convexa. Nótese que el mínimo sobre un lado convexo, que es una cara k en el nivel $\ell = 2$, viene dado por:

$$f_k^* = \frac{A_{ii} - (A_{ii} - A_{ij})^2}{A_{ii} + A_{jj} - 2A_{ij}}. \qquad (14)$$

A. Recorrido del grafo de caras, disminuyendo k

El algoritmo 1 se basa en un recorrido del grafo de caras visitando las caras de mayor a menor indice k, lo que no implica un recorrido por niveles del grafo. Trabaja con una lista global de marcadores Marc_k

- Marc_k=0: la cara k no se ha chequeado,
- Marc_k=1: la cara k se ha chequeado y
- Marc_k=2: no hay que chequear la cara k, ni ninguna de sus sub-caras $\Delta_m \subset \Delta_k$.

Si f es monótona en la cara k, podemos marcar algunas de sus sub-caras como chequeadas [13]. Como todas las subcaras de una cara copositiva son copositivas, encontrar una cara positiva reduce el número de nodos a evaluar en el grafo de caras. Si todas las caras son copositivas entonces la matriz A también lo es. Por lo tanto, la eficiencia del Algoritmo 1 depende de los niveles del grafo en los que se detecten caras copositivas. La existencia de caras copositivas en mayores valores de ℓ reducen el número de subcaras a evaluar.

Algoritmo 1 Copos. decrementando k (CDK)

Entrada: A: matriz simétrica de $n \times n$.
if A es positiva o A es PSD **then**
  return A es copositiva
Evaluar los vértices (A_{ii}), el centroide y los mínimos de los lados convexos de Δ (14), generando G (Def. 2)
if se ha encontrado un punto negativo **then**
  return A no es copositiva
$\text{Marc}_k = 0$ para las caras k no chequeadas
$k = 2^n - 1$
while $k > 2$ **do**
  if $\text{Marc}_k = 0$ **then**
    if $A_k \geq 0$ **then**
      Marc_m=2 para $\Delta_m \subset \Delta_k$
      break
    if $\exists i$, $D_{\ell i}^T A_k > 0$, i.e. f monótona en la dirección i **then**
      Marc_m=1 para $\Delta_m \subset \Delta_k$ que no tengan $x_i = 0$
    else
      if G_k es completo **then** ▶ Def. 2
        if H_k es semidefinida positiva **then**
          Encontrar $x_k^* \in \Delta_k$ según (7)
          if $x^{*T} A_k x^* < 0$ **then**
            return A no es copositiva
          else
            Marc_m=2 para $\Delta_m \subset \Delta_k$
  $k = k$-1
return A es copositiva

La ventaja del algoritmo 1 es que solo hay que almacenar un byte con el marcador Marc_k para cada cara. El inconveniente es que el número de caras se incrementa exponencialmente con n y una cara suele tener varias caras padre en niveles superiores que pueden ser o no copositivas. Por lo tanto, una cara puede ser visitada varias veces para marcarla con Marc_k=2.

B. Recorrido del grafo de mayor a menor nivel

La idea del Algoritmo 2 es la de inspeccionar solo las caras que pueden ser interesantes en cada nivel. En este algoritmo, solo se evalúan las caras no eli-

minadas de la búsqueda de un punto negativo de f en Δ en cada nivel. Para ello, se usan dos listas $\mathcal{L}_\ell$ y $\mathcal{L}_{\ell-1}$ que guardan las caras en nivel ℓ y nivel ℓ-1, respectivamente.

Algoritmo 2 Copos. de arriba hacia abajo (CDown)

Entrada: A: matriz simétrica de $n \times n$.
if A es positiva o A es PSD **then**
  return A es copositiva
Evaluar los vértices (A_{ii}), el centroide y los mínimos de los lados convexos de Δ (14), generando G (Def. 2)
if Encontrado un punto negativo **then**
  return A no es copositiva
$\mathcal{L}_\ell = \{2^n - 1\}$, $\mathcal{R} = \emptyset$, $\ell = n$
while $\ell > 2$ **do**
  $\mathcal{L}_{\ell-1} = \emptyset$, $\mathcal{N} = \emptyset$
  while $\mathcal{L}_\ell \neq \emptyset$ **do**
    Extrae k de $\mathcal{L}_\ell$
    if $A_k \geq 0$ **then**
      Guarda k en $\mathcal{R}$
      continue
    if $\exists i$, $D_{\ell i}^T A_k > 0$, i.e. f monótona en la dirección i **then**
      Guarda m de faceta $\Delta_m \subset \Delta_k$ y $x_i = 0$ en $\mathcal{L}_{\ell-1}$
      Guarda m de faceta $\Delta_m \subset \Delta_k$ y $x_i = 1$ en $\mathcal{N}$
    else
      if G_k es completo **then** ▶ Def. 2
        if H_k es semidefinida positiva **then**
          Encuentra $x_k^* \in \Delta_k$ según (7)
          if $x^{*T} A_k x^* < 0$ **then**
            return A no es copositiva
          else
            Guarda k en $\mathcal{R}$
      else
        Guarda todas las facetas $\Delta_m \subset \Delta_k$ en $\mathcal{L}_{\ell-1}$
  $\mathcal{L}_{\ell-1}$=$\mathcal{L}_{\ell-1} \setminus \mathcal{N}$
  Elimina de $\mathcal{L}_{\ell-1}$ las sub-caras $m : \Delta_m \subseteq \Delta_k, k \in \mathcal{R}$
  $\ell = \ell - 1$
return A es copositiva

Hay una lista global $\mathcal{R}$, que almacena los valores k de las caras con un mínimo interior positivo o que generen una matriz A_k positiva. Ademas, hay una lista $\mathcal{N}$ que almacena las facetas que no pueden tener un mínimo debido a que f es monótona en ellas. De esta forma ninguna sub-cara $k \in \mathcal{R}$ o faceta $k \in \mathcal{N}$ tiene que incluirse en la próxima lista $\mathcal{L}_{\ell-1}$.

En el peor caso, el tamaño de la lista $\mathcal{L}_\ell$ puede ser todavía muy grande para valores altos de la dimensión n. Ademas, la gestión de las listas incrementa la sobrecarga computacional del algoritmo.

C. Recorrido del grafo de menor a mayor nivel

El algoritmo mostrado en [14] busca una solución de (2) subiendo niveles en el grafo de caras para encontrar los cliques máximos del grafo de lados convexos asociados a la matriz A (ver Def. 2).

Para el test de copositividad (ver eq. (3)), los cliques (todos los lados en el grafo G) se visitan de menor a mayor nivel, posiblemente terminando en la evaluación de un(os) clique(s) máximo(s). En cada clique con una H_k que es PSD se evalúa su mínimo para ver si es negativo. En ese caso, el algoritmo 3 termina, ya que la matriz A no es copositiva. Todos los cliques máximos deben ser evaluados para matrices copositivas.

Existen numerosos algoritmos para encontrar los cliques máximos [16] y versiones paralelas [17], [18], [19], [20], [21], [22]. Puede ser interesante adaptar estos algoritmos para resolver (3).

Algoritmo 3 Copos. de abajo hacia arriba (CUp)

Entrada: A: matriz simétrica de $n \times n$.
if A is positiva o A es PSD **then**
  return A es copositiva
Evaluar los vértices (A_{ii}), el centroide y los mínimos de los lados convexos de Δ (14). Los lados convexos se guarda en generando G (Def. 2)
if Encontrado un punto negativo **then**
  return A no es copositiva
$\ell = 2$ y $\mathcal{L}_2$ incluye todos los lados convexos
while $\ell < n - 1$ **do**
  $\mathcal{L}_{\ell+1} = \emptyset$
  while $\mathcal{L}_\ell \neq \emptyset$ **do**
    Extrae m de $\mathcal{L}_\ell$
    for cada vértice $e_j \notin \Delta_m$ **do**
      Añade el vértice e_j a Δ_m generando Δ_k
      if $k \notin \mathcal{L}_{\ell+1}$ **and** G_k es completo **then**
        if H_k es semidefinida positiva **then**
          Encuentra $x_k^* \in \Delta_k$ según (7)
          if $x^{*T} A_k x^* < 0$ **then**
            return A no es copositiva
          Añade k a $\mathcal{L}_{\ell+1}$
  $\ell = \ell + 1$
return A es copositiva

IV. Evaluaciones numéricas

En la evaluación se usan instancias de matrices copositivas de pocas dimensiones tomadas de la literatura para obtener los resultados de los algoritmos descritos anteriormente. La primera instancia

$$A = \begin{pmatrix} 2 & -1 & -1 \\ -1 & 2 & -1 \\ -1 & -1 & 2 \end{pmatrix}$$

sirve como ilustración gráfica de un algoritmo de BBE en [11]. Nuestras implementaciones terminan tras verificar que A es PSD y por lo tanto es copositiva. El algoritmo en [14] primero determina que todos los lados son convexos con un valor mínimo de 0.5, y luego identifica el mínimo con un valor de 0 en el centroide.

El segundo ejemplo es la matriz de Horn, [23].

$$A = \begin{pmatrix} 1 & -1 & 1 & 1 & -1 \\ -1 & 1 & -1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 \\ 1 & 1 & -1 & 1 & -1 \\ -1 & 1 & 1 & -1 & 1 \end{pmatrix}$$

Esta es una matriz copositiva de dimensión $n = 5$, que se corresponde con la figura 1. El algoritmo 1 (CDK) visita todas las caras en 0.01 s. detectando monotonicidad, sin necesitar una descomposición de Cholesky. El algoritmo 2 (CDown) visita todas las caras por nivel en 0.02 s. El algoritmo 3 (CUp) detecta que ninguna de las caras en nivel $\ell = 3$ representa un grafo convexo en 0.05 s. y finaliza la búsqueda del mínimo sin visitar las caras de niveles superiores.

También hemos usado varias instancias del problema del Clique máximo (ver el apéndice), aunque existen métodos más eficaces para resolverlos. Sea $\mathbf{1}$ el vector con todos los elementos a uno en la dimensión apropiada, ω el número Clique de un grafo con

matriz de adyacencia Adj. El objetivo de este problema de programación copositiva es encontrar el menor valor de t tal que $(t-1)\mathbf{1}\mathbf{1}^T - tAdj$ es copositiva, i.e. el número Clique es

$$\omega = \text{mín}\{t : [(t-1)\mathbf{1}\mathbf{1}^T - tAdj] \text{ es copositiva}\}. \quad (15)$$

En el primer ejemplo, con una dimensión $n = 8$, la matriz es copositiva para $t = 4$. En este caso, una version preliminar del algoritmo BBE [11] requiere 359,000 evaluaciones de la función objetivo para determinar la ϵ-copositividad con $\epsilon = 10^{-6}$. El algoritmo de [14] realiza una búsqueda de un mínimo hasta el nivel $\ell = 5$. Nuestros algoritmos detectan directamente que A es PSD y por lo tanto copositiva.

Para valores de $t = 2, 3$, el centroide tiene un valor negativo. Nuestros algoritmos lo detectan y ahorran la búsqueda en las caras.

Hemos tomado tres problemas de clique máximo del reto de DIMACS con dimensiones de $n = 14, 16$ y $n = 28$. Las tablas I a III muestran los siguientes resultados: `mono`: número de veces que se ha demostrado que f es monótona en una cara; `allpos`: A_k es completamente positiva; `Chol`: número de descomposiciones de Cholesky para determinar si una matriz es PSD; `nivel`: donde se ha encontrado un punto con un valor negativo; `tiempo`: tiempo de ejecución, `cp`: si se ha demostrado copositividad. Los algoritmos se han implementado en Matlab R2016 y se han ejecutado en un PC con una CPU i5.

Algoritmo CDK recorre la lista de las 2^n-1 caras y las marca o no dependiendo de la detección de monotonía y la existencia caras en un nivel superior con un mínimo interior positivo o una matriz completamente positiva. Se puede observar que siempre que se hace una descomposición de Cholesky, H_k resulta ser PSD y por lo tanto se evalúa el mínimo.

TABLA I
Resultados para Algoritmo 1, CDK.

n	t	mono	allpos	Chol	#f	nivel	tiempo	cp
14	3	16	0	2	2	5	0,16s	0
14	4	16	0	2	2	5	0,16s	0
14	5	209	105	31	31	.	0,78s	1
16	6	0	0	2	2	8	0,55s	0
16	7	0	0	2	2	8	0,55s	0
16	8	17	3	46	46	.	2,98s	1
28	3	13441	52	2	2	4	6.707s	0
28	4	50142	370	106	106	.	11.898s	1

La instancia más grande que resolvemos aquí es el problema Johnson8-2-4 [24], con n=28 y un clique máximo de t=4. Esto significa que si cada marca Marc_k de una cara k consume un byte, se requieren 2^{28} bytes, i.e. 0.5 GB. Para t=3 la matriz no es copositiva y el Algoritmo CDK necesita 1.9 horas para detectar un punto (mínimo interior) negativo en el nivel 4. Para t=4 la matriz es copositiva y el algoritmo requiere 3 horas y 20 minutos para recorrer toda la lista. El Algoritmo CDown intenta no correr toda la lista ya que las caras se visitan por nivel. No todas las caras de un nivel son evaluados gracias a los conjuntos de caras $\mathcal{R}$ y $\mathcal{N}$. Se puede observar que este algoritmo pierde mucho tiempo en la gestión de la listas por niveles. Solo el nivel 14 tiene $\binom{28}{14}$=40,116,600 caras. Por este motivo, el tiempo requerido por el algoritmo CDown supera la de algoritmo CDK. El Algoritmo CUp necesitó menos de 2 s.

TABLA II
Resultados para Algoritmo 2, CDown.

n	t	#cara	$\lvert\mathcal{C}\rvert$	mono	allpos	#f	nivel	tiempo	cp
14	3	14,567	0	85	0	2	5	2,53s	0
14	4	14,567	0	111	0	2	5	2,67s	0
14	5	16,053	33	203	3	31	.	3,14s	1
16	6	36,639	0	0	0	2	8	9,71s	0
16	7	36,639	0	0	0	2	8	9,71s	0
16	8	64,226	48	17	3	46	.	15,5s	1
28	3	.	.	.	.	.	.	>24 h	0
28	4	.	.	.	.	.	.	>24 h	1

Los resultados en [14] sobre matrices generadas aleatoriamente para n=10, 30, 50, 100, 200, 500, 1000 y 1500, y densidades de sus grafos de lados convexos de 0.25, 0.5 y 0.75 que fueron generadas según los procedimientos descritos en [25], [26], muestran que se pudieron resolver de forma exhaustiva problemas de densidad 0.25 hasta n=500, de densidad 0.5 hasta n=200 y de densidad 0.75 hasta n=100. Esto indica que una búsqueda de abajo hacia arriba es interesante para matrices con grafos de lados convexos poco densos y suponemos que una búsqueda de arriba hacia abajo podría ser mas interesante para matrices con grafos de lados convexos más densos.

TABLA III
Resultados para Algoritmo 3, CUp.

n	t	#cara	#f	nivel	tiempo	cp
14	3	177	73	4	0,10s	0
14	4	207	103	5	0,13s	0
14	5	203	105	6	0,16s	1
16	6	1459	1324	8	0,73s	0
16	7	1459	1324	8	0,76s	0
16	8	1461	1326	8	0,76s	1
28	3	827	422	4	1,26s	0
28	4	931	526	4	1,48s	1

V. Conclusiones

Los algoritmos exhaustivos basados en las caras de un simplex unidad parecen más eficientes que los que realizan una división espacial para demonstrar la copositividad de una matriz. En este trabajo se han estudiado diferentes formas de recorrido del grafo de caras. Uno de los retos computacionales es la gestión de las listas de caras a evaluar. El mejor recorrido parece depender de la densidad del grafo de lados convexos asociado a la matriz.

Agradecimientos

Este trabajo ha sido subvencionado por el Ministerio de Ciencia e Innovación (RTI2018-095993) y financiado en parte por el Fondo Europeo de Desarrollo Regional (ERDF).

Referencias

[1] Immanuel M. Bomze, "On standard quadratic optimization problems," *Journal of Global Optimization*, vol. 13, no. 4, pp. 369–387, Dec 1998.

[2] Harry Markowitz, "Portfolio selection," *The journal of Finance*, vol. 7, no. 1, pp. 77–91, 1952.

[3] J. F. C. Kingman, "A mathematical problem in population genetics," *Mathematical Proceedings of the Cambridge Philosophical Society*, vol. 57, no. 3, pp. 574–582, 1961.

[4] Toshihide Ibaraki and Naoki Katoh, *Resource Allocation Problems: Algorithmic Approaches*, MIT Press, Cambridge, MA, USA, 1988.

[5] Immanuel M. Bomze, "Regularity versus degeneracy in dynamics, games, and optimization: A unified approach to different aspects," *SIAM Review*, vol. 44, no. 3, pp. 394–414, 2002.

[6] Immanuel M. Bomze, "Branch-and-bound approaches to standard quadratic optimization problems," *Journal of Global Optimization*, vol. 22, no. 1, pp. 17–37, Jan 2002.

[7] Stefan Bundfuss and Mirjam Dur, "An adaptive linear approximation algorithm for copositive programs," *SIAM Journal on Optimization*, vol. 20, no. 1, pp. 30–53, 2009.

[8] J. Gondzio and E. A. Yildirim, "Global solutions of nonconvex standard quadratic programs via mixed integer linear programming reformulations," Tech. Rep. ERGO-18-022, School of Mathematics, The University of Edinburgh, 2018.

[9] G. Liuzzi, M. Locatelli, and V. Piccialli, "A new branch-and-bound algorithm for standard quadratic programming problems," *Optimization Methods and Software*, vol. 34, no. 1, pp. 79–97, 2019.

[10] Katta G. Murty and Santosh N. Kabadi, "Some NP-complete problems in quadratic and nonlinear programming," *Mathematical Programming*, vol. 39, no. 2, pp. 117–129, 1987.

[11] S. Bundfuss and M. Dür, "Algorithmic copositivity detection by simplicial partition," *Linear Algebra and its Applications*, vol. 428, no. 7, pp. 1511–1523, 2008.

[12] J. Žilinskas and M. Dür, "Depth-first simplicial partition for copositivity detection, with an application to maxclique," *Optimization Methods and Software*, vol. 26, no. 3, pp. 499–510, 2011.

[13] E. M. T. Hendrix, J. M. G. Salmerón, and L. G. Casado, "On function monotonicity in simplicial branch and bound," *AIP Conference Proceedings*, vol. 2070, no. 1, pp. 020007, 2019.

[14] Andrea Scozzari and Fabio Tardella, "A clique algorithm for standard quadratic programming," *Discrete Applied Mathematics*, vol. 156, no. 13, pp. 2439 – 2448, 2008.

[15] J.M.G. Salmerón, L.G. Casado, and E.M.T. Hendrix, "Paralelización de la detección de una matriz copositiva mediante la evaluación de las facetas de un simplex unidad," in *Libro de Abstracts de las Jornadas SARTECO 2018*. July 2018, pp. 77–82, Universidad de Zaragoza.

[16] Qinghua Wu and Jin-Kao Hao, "A review on algorithms for maximum clique problems," *European Journal of Operational Research*, vol. 242, no. 3, pp. 693 – 709, 2015.

[17] Ryan A. Rossi, David F. Gleich, Assefaw H. Gebremedhin, and Md. Mostofa Ali Patwary, "Fast maximum clique algorithms for large graphs," in *Proceedings of the 23rd International Conference on World Wide Web*, New York, NY, USA, 2014, WWW '14 Companion, pp. 365–366, ACM.

[18] Matjaž Depolli, Janez Konc, Kati Rozman, Roman Trobec, and Dušanka Janežič, "Exact parallel maximum clique algorithm for general and protein graphs," *Journal of Chemical Information and Modeling*, vol. 53, no. 9, pp. 2217–2228, 2013.

[19] Ciaran McCreesh and Patrick Prosser, "The shape of the search tree for the maximum clique problem and the implications for parallel branch and bound," *ACM Trans. Parallel Comput.*, vol. 2, no. 1, pp. 8:1–8:27, Apr. 2015.

[20] Boyi Hou, Zhuo Wang, Qun Chen, Bo Suo, Chao Fang, Zhanhuai Li, and Zachary G. Ives, "Efficient maximal clique enumeration over graph data," *Data Science and Engineering*, vol. 1, no. 4, pp. 219–230, Dec 2016.

[21] Pablo San Segundo, Jorge Artieda, and Darren Strash, "Efficiently enumerating all maximal cliques with bit-parallelism," *Computers & Operations Research*, vol. 92, pp. 37 – 46, 2018.

[22] A. Das, S. Sanei-Mehri, and S. Tirthapura, "Shared-memory parallel maximal clique enumeration," in *2018 IEEE 25th International Conference on High Performance Computing (HiPC)*, 2018, pp. 62–71.

[23] Marshall Hall and Morris Newman, "Copositive and completely positive quadratic forms," *Mathematical Proceedings of the Cambridge Philosophical Society*, vol. 59, no. 2, pp. 329–339, 1963.

[24] "Second DIMCAS implementation challengue," http://archive.dimacs.rutgers.edu/Challenges/.

[25] Immanuel M. Bomze and Etienne De Klerk, "Solving standard quadratic optimization problems via linear, semidefinite and copositive programming," *Journal of Global Optimization*, vol. 24, no. 2, pp. 163–185, 2002.

[26] Ivo Nowak, "A new semidefinite programming bound for indefinite quadratic forms over a simplex," *Journal of Global Optimization*, vol. 14, no. 4, pp. 357–364, Jun 1999.

Appendix: Instancias de matrices

1tc.8.clique, $n = 8$, oeis.org/A265032/a265032.html.

$$Adj = \begin{pmatrix} 0&1&1&1&1&1&1&1\\ 1&0&0&1&0&1&1&1\\ 1&0&0&1&0&1&1&1\\ 1&1&1&0&1&0&0&1\\ 1&0&0&1&0&1&1&1\\ 1&1&1&0&1&0&0&1\\ 1&1&1&0&1&0&0&1\\ 1&1&1&1&1&1&1&0 \end{pmatrix},$$

$A = (t-1)\mathbf{1}\mathbf{1}^T - tAdj$ es copositiva para $t = 4$.

Brock14, $n = 14$, The Second DIMACS Implementation Challenge.
Generado con *grahgen -g14 -c5 -p0.5 -d0 -s0.*

$$Adj = \begin{pmatrix} 0&1&1&0&1&1&0&1&0&1&0&1&0&1\\ 1&0&1&1&1&0&0&1&0&1&0&1&0&0\\ 1&1&0&0&1&1&0&0&1&0&1&1&1&1\\ 0&1&0&0&1&0&1&0&0&1&1&1&0&1\\ 1&1&1&1&0&1&1&1&1&0&0&0&1&0\\ 1&0&1&0&1&0&1&1&0&1&0&1&1&1\\ 0&0&0&1&1&1&0&0&1&1&0&0&1&1\\ 1&1&0&0&1&1&0&0&0&0&1&0&1&1\\ 0&0&1&0&1&0&1&0&0&1&1&0&1&1\\ 1&1&0&1&0&1&1&0&1&0&1&0&1&1\\ 0&0&1&1&0&0&0&1&1&1&0&1&0&0\\ 1&1&1&1&0&1&0&0&0&0&1&0&0&1\\ 0&0&1&0&1&1&1&1&1&1&0&0&0&1\\ 1&0&1&1&0&1&1&1&1&1&0&1&1&0 \end{pmatrix},$$

$A = (t-1)\mathbf{1}\mathbf{1}^T - tAdj$ es copositiva para $t = 5$.

$n = 16$, The Second DIMACS Implementation Challenge.

$$Adj = \begin{pmatrix} 0&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1\\ 1&0&0&1&0&1&1&1&1&1&1&1&1&1&1&1\\ 1&0&0&1&0&1&1&1&0&1&1&1&1&1&1&1\\ 1&1&1&0&1&0&0&1&1&0&1&1&1&1&1&1\\ 1&0&0&1&0&1&1&1&0&1&1&1&1&1&1&1\\ 1&1&1&0&1&0&0&1&1&0&0&1&1&1&1&1\\ 1&1&1&0&1&0&0&1&1&0&0&1&0&1&1&1\\ 1&1&1&1&1&1&1&0&1&1&1&0&1&0&0&1\\ 1&1&0&1&0&1&1&1&0&1&1&1&1&1&1&1\\ 1&1&1&0&1&0&0&1&1&0&0&1&0&1&1&1\\ 1&1&1&1&1&0&0&1&1&0&0&1&0&1&1&1\\ 1&1&1&1&1&1&1&0&1&1&1&0&1&0&0&1\\ 1&1&1&1&1&1&1&0&1&1&0&0&1&0&1&1\\ 1&1&1&1&1&1&1&1&0&1&1&1&0&1&0&0\\ 1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&0 \end{pmatrix},$$

$A = (t-1)\mathbf{1}\mathbf{1}^T - tAdj$ es copositive para $t = 8$.

Nueva implementación paralela en GPUs del algoritmo *pLSA* para desmezclado de imágenes hiperespectrales

Jose A. Gallardo[1], Mercedes E. Paoletti[1], Juan M. Haut[1], Javier Plaza[1] y Antonio Plaza[1]

Resumen— **Los algoritmos de desmezclado son una parte importante dentro del campo del análisis de imágenes hiperespectrales de la superficie terrestre. Estas técnicas permiten extraer, por cada píxel, una firma espectral pura (*endmember*) y su material mas abundante. En este trabajo, proponemos un nuevo método para realizar este proceso de forma eficiente. El método propuesto, hace uso del algoritmo *probabilistic latent semantic analysis* (pLSA) para inferir los *endmembers* y las abundancias de cada píxel de forma simultánea en el espacio latente de la imagen. Los métodos basados en procesos estadísticos, como pLSA, generan una carga computacional demasiado elevada como para ser una solución de uso diario en sus versiones serie. Por este motivo, proponemos también una versión que explota las ventajas del paralelismo a nivel GPU mediante el *framework* CUDA. Para demostrar la robustez de la versión propuesta, se proporciona experimentación realizada sobre dos imágenes hiperespectrales, comparando la implementación CUDA con respecto a la versión serie del algoritmo.**

Palabras clave— **GPU, CUDA, desmezclado, hiperespectral**

I. Introducción

En la actualidad, el tratamiento y análisis de imágenes hiperespectrales es uno de los ámbitos mas en auge dentro del campo de la teledetección [1, 2]. Esto se debe a la multitud de posibilidades que ofrece el análisis de esta clase de imágenes, desde la resolución de problemas de clasificación [3–5], hasta la creación de mapas de terreno de alta precisión [6, 7], existen multitud de campos donde el uso de técnicas de teledetección resulta extremadamente útil. Dentro de estas áreas se incluye el desmezclado hiperespectral[8, 9], que separa la imagen en dos componentes principales: los *endmembers* y las abundancias. Mientras que los *endmembers* representan la firma espectral del componente mas abundante del píxel, las abundancias nos indican (de forma proporcional) la cantidad de cada muestra existente en un píxel.

El estado del arte actual ofrece algunos métodos capaces de realizar el desmezclado con menor o mayor precisión. Por ejemplo, son muy populares las técnicas basadas en soluciones geométricas, como por ejemplo, el análisis de vértices principales (VCA) [10], que utiliza geometría convexa para identificar los *endmembers* como puntos extremos en la imagen, o el método de análisis del mínimo volumen (MVSA), que intenta englobar todos los puntos de la imagen mediante el *simplex* de mínimo volumen, utilizando restricciones para mejorar la fiabilidad de la solución obtenida. Además de estas soluciones geométricas, existen otras soluciones basadas en métodos estadísticos, donde las abundancias y los *endmembers* son modelados mediante una distribución estadística [11, 12].

Estas soluciones estadísticas se sitúan como una alternativa mas robusta cuando la escena a analizar presenta un alto grado de mezcla espectral, pues permiten obtener una perspectiva mas profunda del modelo de datos [8]. Existen publicaciones recientes que demuestran las ventajas inherentes al uso de soluciones basadas en representaciones de datos [13], siendo los métodos probabilísticos un campo muy prometedor dentro del análisis de imágenes hiperespectrales [14–16]. Concretamente, las soluciones basadas en modelos generativos probabilísticos se han convertido en una herramienta de representación de datos mediante el uso de patrones semánticos latentes [17]. Debido a esto, estos modelos aportan grandes ventajas con respecto a otros modelos de desmezclado hiperespectral [18]. Específicamente, en este trabajo presentamos un modelo probabilístico llamado *dual-depth-pLSA* (DEpLSA), basado en el algoritmo pLSA [19] tradicional, capaz de demezclar de forma eficiente conjuntos de datos hiperespectrales. Se muestran las ventajas de la implementación del algoritmo DEpLSA con respecto al resto de métodos de desmezclado. Sin embargo, a pesar de sus ventajas a la hora de realizar el desmezclado, existe un problema a la hora de ejecutar este algoritmo: el gran coste computacional inherente a los modelos probabilísticos basados en el aprendizaje bayesiano [20, 21].

A pesar de que existen en el estado del arte soluciones probabilísticas que explotan las ventajas de la computación paralela [19, 22, 23], la complejidad de este algoritmo, unida a la complejidad de los datos hiperespectrales, lo convierten en un candidato ideal sobre el que desarrollar y probar soluciones paralelas.

La implementación paralela propuesta utiliza coprocesadores gráficos de propósito general (GPUs) para resolver este problema. Teniendo en cuenta las operaciones matriciales que realiza el algoritmo de optimización empleado en pLSA: el algoritmo de expectación/maximización [18], es posible desarrollar una solución basada en Nvidia CUDA capaz de paralelizar estos cálculos de forma muy eficiente.

En la sección de experimentos se proporciona una comparativa de nuestra versión desarrolada en CU-

[1]Hyperspectral Computing Laboratory, Department of Technology of Computers and Communications, Escuela Politécnica, University of Extremadura, 10003 Cáceres, Spain (e-mail: mpaoletti@unex.es; juanmariohaut@unex.es; jplaza@unex.es; aplaza@unex.es)

DA con respecto a las implementaciones serie y paralela en CPU del algoritmo DepLSA. Para ello, se han utiliando varias imágenes hiperespectrales reales.

II. DEpLSA para desmezclado hiperespectral

DEpLSA es un método estadístico de desmezclado basado en el concepto de la semántica latente [24], donde la solución al problema de desmezclado se obtiene estimando abundancias y *endmembers* de acuerdo a variables del espacio latente. Concretamente, podemos definirlo como un proceso de desmezclado semi-generativo que, considerando las variables latentes z y z', es capaz de realizar el demezclado. La Fig. 1(a) representa este modelo de manera gráfica. En la figura, representamos los píxeles mediante d y las variables latentes mediante z, que representa el espacio profundo (utilizado para generar representaciones semánticas de la imagen) y z', utilizado para inferir los *endmembers* y las abundancias. Además, tenemos las variables M y N, que denotan el numero total de píxeles y bandas espectrales de la imagen hiperespectral. Además, entran en juego dos regularizadores: r_d y r_z, para asegurar la dispersión de los datos en los modelos probabilísticos $p(z|d)$ y $p(w|z)$, que describen las abundancias y los *endmembers*, respectivamente.

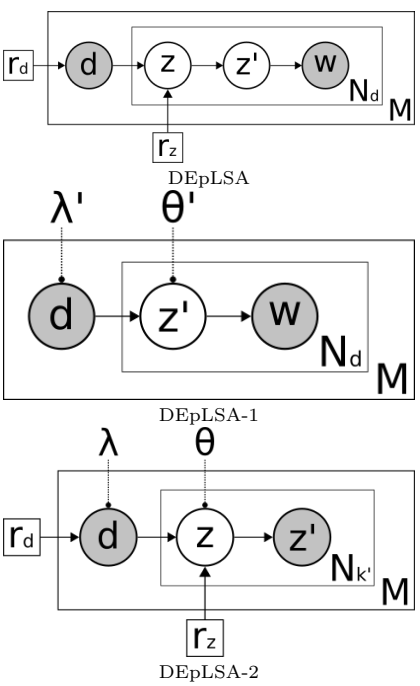


Fig. 1
Pipeline de datos del algoritmo DEpLSA (a), y de los dos pasos que lo componen (b y c).

La principal ventaja del algoritmo DEpLSA consiste en el uso de un primer espacio latente reducido z', del cual se inferirá en un segundo paso (el espacio de *endmembers* y abundancias de la imagen). Esta doble reducción mediante variables latentes genera un gran coste computacional, por lo que el algoritmo realiza dos procesos de modelado:

1. El paso 1 realiza una reducción al espacio profundo K' utilizando como entrada la imagen hiperespectral [ver Fig. 1(b)].
2. El paso 2 utiliza la salida del paso 1 para estimar el espacio de *endmembers* (K) y abundancias de la imagen [ver Fig. 1(c)].

La fuerte dependencia del proceso de expectación/maximización de este algoritmo deriva en una fuerte explotación de los recursos de cómputo masivamente paralelos. En la siguiente sección se describe una implementación paralela del algoritmo en GPUs.

III. Implementación paralela de DePLSA utilizando Nvidia CUDA

Esta sección detalla el funcionamiento interno de nuestra implementación GPU del algoritmo DEpLSA. El objetivo principal de la implementación es minimizar el tiempo de cómputo del algoritmo, sin descuidar la optimización de la gestión de memoria GPU y el tiempo de comunicación y entrada/salida de datos.

Como se mencionó anteriormente, el algoritmo de optimización expectación/maximización supone la mayor parte de la carga computacional, siendo este algoritmo el principal objetivo de la paralelización. Puesto que todas las operaciones llevadas a cabo por este algoritmo son realizadas sobre matrices que contienen datos probabilísticos, es posible implementar de forma eficiente una versión paralela en el que cada índice de estas matrices es calculado de forma paralela en cada núcleo de la GPU, distribuyéndose estos en tiempo real durante la ejecución del algoritmo. Esto es posible gracias a la gestión de hilos de CUDA, que genera una malla de hilos de 1, 2 ó 3 dimensiones en el que cada elemento de la malla es a su vez un conjunto de hilos llamado bloque. Estos bloques, al igual que las mallas, pueden ser uni-, bi- o tridimensionales. Esta dimensionalidad espacial de bloques y mallas se parametriza para cada *kernel*. En este sentido, nuestra implementación aprovecha este principio para garantizar la atomicidad del conjunto de operaciones. La Fig. 2 muestra un ejemplo gráfico de este paradigma de gestión de hilos.

A. Optimización de la gestión y accesos a memoria

Dentro de DEpLSA existen tres estructuras de datos sobre las que se realiza el computo: θ (*endmembers*), λ (abundancias), y la imagen original. Para evitar un uso constante de las operaciones de entrada/salida, se realiza una reserva de memoria de estas matrices a GPU al inicio del algoritmo. Esto permite no realizar mas operaciones de transferencia de datos hasta el envío de las soluciones al dispositivo controlador.

B. Implementación paralela del paso de Expectación

El paso de expectación trata de generar un nuevo espacio de variables latentes basado en el espacio de *endmembers* (θ) y abundancias (λ) actual. El espacio latente generado se almacena en un estructura tridimensional llamada p. Es sencillo garantizar la

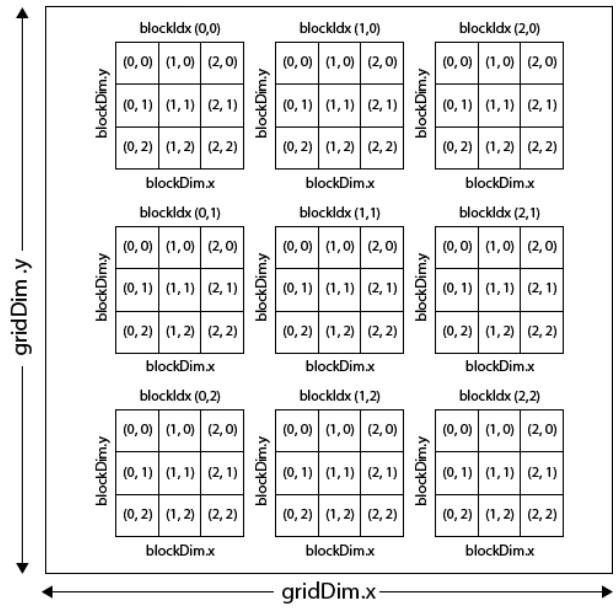


Fig. 2
Representación gráfica de la gestión de hilos en CUDA.

atomicidad de cada elemento del cubo de datos hiperespectral asignando los hilos del *kernel* de forma que cada hilo siempre se encarga del mismo índice del cubo, almacenando los denominadores comunes en la memoria compartida de la GPU. El Algoritmo 1 muestra en forma de pseudocódigo las operaciones paralelas realizadas en este paso.

Algorithm 1 Kernel GPU del paso de expectación del algoritmo

```
1:  procedure KERNEL
2:      for Bloque en Malla[X,Y] do ▷ En paralelo
3:          den ← 0                 ▷ Memoria compartida
4:          for Hilo en Bloque[Z] do ▷ En paralelo
5:              P[hilo] ← λ[hilo] × θ[hilo]
6:              den[bloque] ← den[bloque] + P[hilo] ▷ Atomicoo
7:              P[hilo] ← P[hilo]/den[bloque]
8:          end for
9:      end for
10: end procedure
```

C. *Implementación paralela del paso de Maximización*

Este paso contiene una serie de peculiaridades que impiden su paralelización en un único *kernel* como ocurre en el paso anterior. En este caso, es necesario utilizar una cadena de *kernels* para garantizar la atomicidad, dividiendo el proceso de cómputo en tres subprocesos atómicos. En concreto, la actualización de los valores de θ se divide en tres *kernels* encargados de realizar cada una de las partes del cálculo correspondiente:

1. El primer paso actualiza el numerador de la fracción, como se muestra en el Algoritmo 2.
2. El segundo paso realiza las reducciones del denominador; esto lo realiza el *kernel* del Algoritmo 3.
3. El último paso utiliza las salidas de los dos anteriores para realizar la división y asignarla en *theta* como se ve en el Algoritmo 4.

En este caso las dependencias a nivel de memoria compartida no se solventan a nivel de bloque, por tanto es necesario buscar una solución alternativa que preserve la atomicidad, dada la aparición de un bucle *for* en el Algoritmo 2.

Algorithm 2 Calculo del numerador de la fracción de *endmembers* (θ)

```
1:  procedure KERNEL
2:      for Bloque en Malla[M,K] do ▷ En paralelo
3:          for Hilo en Bloque[1024] do ▷ En paralelo
4:              θ[bloque] ← 0
5:              for paso en pasos do
6:                  aux ← (X[paso] × P[paso]) − regulador
7:                  if aux > 0 then
8:                      θ[bloque] ← aux ▷ Atomico
9:                  end if
10:             end for
11:         end for
12:     end for
13: end procedure
```

Algorithm 3 Cálculo del denominador de la fracción de *endmembers* (θ)

```
1:  procedure KERNEL
2:      for Bloque en Malla[K] do ▷ En paralelo
3:          for Hilo en Bloque[M] do ▷ En paralelo
4:              den[bloque.z] ← den[bloque.z]+θ[hilo]
5:          end for
6:      end for
7:  end procedure
```

Algorithm 4 Cálculo de la fracción de *endmembers* (θ)

```
1:  procedure KERNEL
2:      for Bloque en Malla[M, K] do ▷ En paralelo
3:          if den[bloque.z] ≠ 0 then
4:              θ[hilo] ← θ[hilo]/den[bloque.z]
5:          else
6:              θ[hilo] ← 1/M
7:          end if
8:      end for
9:  end procedure
```

Una vez terminado el cálculo de los *endmemebers*, el paso de maximización se encarga de actualizar la matriz de abundancias λ de forma similar al proceso anterior. Sin embargo, al depender en este caso

el cálculo del denominador común de los elementos de un bloque, es posible implementar un único *kernel* que se encargue de un único elemento de λ. Este *kernel* utiliza bloques de tamaño $N \times K$, cada uno gestionando M hilos. Se proporciona también un pseudocódigo de este proceso en el algoritmo 5

Algorithm 5 Calculo de la fracción de abundancias (λ)

```
1:  procedure Kernel
2:      for Bloque in Malla[X,Z] do ▷ En paralelo
3:          den ← 0                 ▷ Memoria compartida
4:          for Thread in Bloque[Y] do          ▷ En paralelo
5:              aux ← (X[paso] × P[paso]) − regulador
6:              if aux > 0 then
7:                  λ[bloque] ← aux             ▷ Atómico
8:              end if
9:              den ← den + X[hilo]             ▷ Atómico
10:             if den ≠ 0 then
11:                 λ[hilo] ← θ[hilo]/den
12:             else
13:                 λ[hilo] ← 1/K
14:             end if
15:         end for
16:     end for
17: end procedure
```

IV. Experimentación

A. Entorno de pruebas

Para demostrar la fiabilidad de la implementación propuesta, se ha realizado una batería de pruebas sobre una versión serie (utilizada como base para el cálculo del *speedup*), una versión optimizada (con los *flags -O3* y *-xAVX*) y la implementación CUDA propuesta. Las versiones han sido ejecutadas en un entorno de cómputo compuesto por un procesador Intel® Core™i7-6700K de 6ª generación con 4/8 núcleos físicos/lógicos, cuya frecuencia puede llegar hasta 4.20 Ghz en modo turbo, acompañado de 40 GB de memoria DDR4 a 2400 Mhz de frecuencia y un soporte de almacenamiento Toshiba DT01ACA HDD a 7200RPM y 2TB de capacidad. Para demostrar la escalabilidad de la implementación se han considerado dos entornos GPU:

1. La primera ejecución se realiza sobre una NVIDIA GeForce GTX 1080, con 2560 núcleos CUDA, 8GB GDDR5X de memoria de vídeo y 10 Gbps de frecuencia (a partir de ahora denominada GPU1).
2. La segunda ejecución se realiza sobre una GPU Tesla P100, compuesta por 3584 núcleos CUDA, 16GB HBM2 de memoria de vídeo y 12 Gbps de frecuencia (a partir de ahora denominada GPU2).

B. Imágenes hiperespectrales

Para demostrar que la escalabilidad de la implementación aumenta a la par que lo hace el tamaño de los datos de entrada, se han tenido en cuenta dos imágenes hiperespectrales de diferente tamaño y complejidad espectral:

1. La primera imagen es la denominada Samson, que contiene 952×952 píxeles y 156 bandas. Para desmezclado hiperespectral se considera una región de $95 \times 95 \times 156$ píxeles con tres endmembers.
2. La segunda imagen es la denominada Cuprite, un conjunto de datos complejo, compuesto por doce endmembers, que comprende una zona de 250×190 píxeles con 224 bandas. Tras el proceso de eliminación de bandas ruidosas, consideramos un cubo de datos de dimensiones $250 \times 190 \times 188$.

C. Experimentación y resultados

En esta sección se estudia el rendimiento a nivel de cómputo de la solución propuesta. La Tabla I muestra los resultados de ejecución de las versiones serie, optimizada y GPU mencionadas en la sección anterior para los algoritmos pLSA y DEpLSA. Como se puede observar en la tabla, la versión optimizada proporciona mejoras sobre el código serie. Sin embargo, es la versión GPU la que desarrolla todo el potencial paralelo del algoritmo, siendo este mejor explotado cuanto mas potente es el dispositivo de cómputo (la ejecución a sobre la segunda GPU es tres veces mejor que la de la primera). No se proporcionan resultados de DEpLSA sobre la imagen de Cuprite debido a limitaciones de memoria.

Por ultimo, para demostrar la robustez del cauce de datos de la implementación propuesta, se recogen una serie de métricas de forma gráfica (obtenidas con la herramienta de análisis de implementaciones GPU Nvidia Visual Profiler). En la Fig. 3 podemos observar el cauce de datos y la repartición de los *kernels* dentro de la GPU. El *kernel* encargado de calcular el paso de expectación ocupa la mayor parte del cómputo GPU pues, a diferencia del resto de *kernels* que realizan cálculos sobre matrices, el paso de expectación genera el nuevo conjunto latente tridimensional.

Por otro lado, en Fig. 4 se recogen datos referentes a las transferencias a memoria (se transfieren 22 GB de datos a razón de 4.77 GB/s) y a la ocupación de los *kernels* dentro de la GPU. Se puede apreciar que, durante el cálculo de los *endmembers* y de las abundancias, la GPU utiliza todos sus recursos con gran eficiencia.

V. Conclusiones y trabajo futuro

En este trabajo presentamos una nueva implementación paralela del algoritmo pLSA para desmezclado de imágenes hiperespectrales. La implementación propuesta aprovecha las ventajas inherentes a la computación en plataformas con un gran numero de núcleos (GPUs). Los experimentos realizados sobre dos GPUs respaldan la fiabilidad de la implementación propuesta y sugieren que, a medida que las GPUs vayan adquiriendo mayor capacidad de cómputo, la implementación propuesta seguirá aprovechando esta capacidad en favor de su escalabilidad.

TABLA I

Tiempos de ejecución (en segundos) de las implementaciones serie y paralelas de pLSA y DEpLSA en las imágenes consideradas.

Dataset	K	pLSA	OP-pLSA	GPU1-pLSA	GPU2-pLSA	Speedup Optimizado	Speedup GPU1	Speedup GPU2
Samson	3	160.43	105.95	4.70	2.94	1.51	34.14	54.61
Cuprite	12	5584.27	4622.68	163.66	55.51	1.21	34.12	100.60
Dataset	K	DEpLSA	OP-DEpLSA	GPU1-DEpLSA	GPU2-DEpLSA	Speedup Optimizado	Speedup GPU1	Speedup GPU2
Samson	3	2440.85	3404.03	61.23	21.89	1.03	39.86	111.51
Cuprite	12	26990.26	26791.35	N/A	198.68	1.01	N/A	135.85

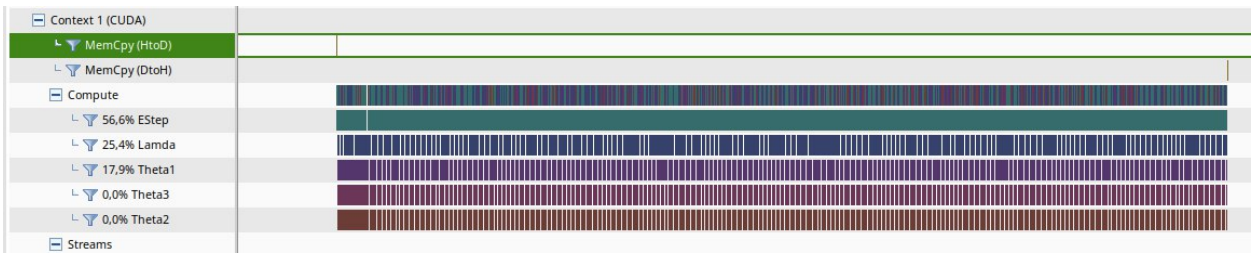
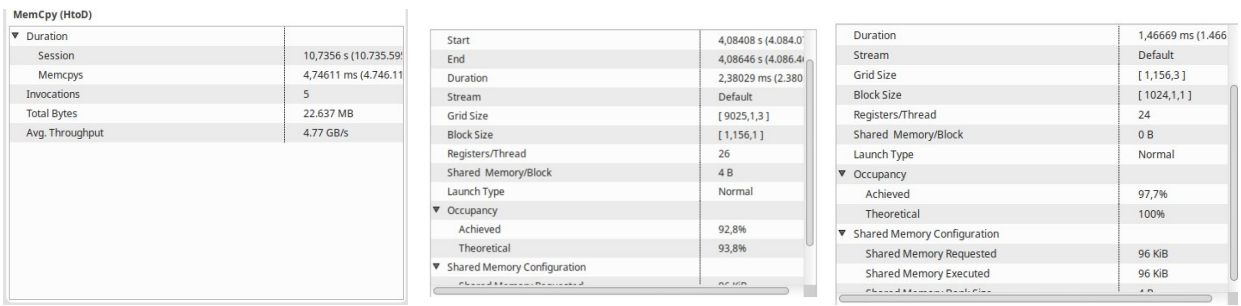


Fig. 3

Representación gráfica del cauce de datos durante la ejecución GPU (obtenido mediante NVidia CUDA Profiler). Se puede apreciar como las trasferencias a memoria son realizadas únicamente al principio y al final del proceso.



(a) Ancho de banda (b) Ocupación Abundancias (c) Ocupación *Endmembers*

Fig. 4

Datos de aprovechamiento de ancho de banda y ocupación (% de núcleos utilizados de GPU) durante el cálculo de abundancias y endmembers.

En el futuro, analizaremos la posibilidad de incorporar funcionalidades multi-GPU que mejorarían aun mas el rendimiento del algoritmo propuesto. También realizaremos implementaciones del algoritmo en otros dispositivos como el *Intel Xeon Phi* y *field programmable gate arrays* (FPGAs).

Agradecimientos

Este trabajo ha sido financiado por el Ministerio de Educación (Resolución de 26 de diciembre de 2014 y de 19 de noviembre de 2015, de la Secretaría de Estado de Educación, Formación Profesional y Universidades, por la que se convocan ayudas para la formación de profesorado universitario, de los subprogramas de Formación y de Movilidad incluidos en el Programa Estatal de Promoción del Talento y su Empleabilidad, en el marco del Plan Estatal de Investigación Científica y Técnica y de Innovación 2013-2016). Este trabajo también ha sido financiado por la Junta de Extremadura (Decreto 14/2018, de 6 de febrero, por el que se establecen las bases reguladoras de las ayudas para la realización de actividades de investigación y desarrollo tecnológico, de divulgación y de transferencia de conocimiento por los Grupos de Investigación de Extremadura, Ref. GR18060). El trabajo también ha sido parcialmente financiado por el programa Horizonte 2020 de la Unión Europea, Research and Innovation Programme, Grant No. 734541 (EOXPOSURE).

Referencias

[1] José M Bioucas-Dias, Antonio Plaza, Gustavo Camps-Valls, Paul Scheunders, Nasser Nasrabadi, and Jocelyn Chanussot, "Hyperspectral remote sensing data analysis and future challenges," *IEEE Geoscience and remote sensing magazine*, vol. 1, no. 2, pp. 6–36, 2013.

[2] Pedram Ghamisi, Naoto Yokoya, Jun Li, Wenzhi Liao, Sicong Liu, Javier Plaza, Behnood Rasti, and Antonio Plaza, "Advances in hyperspectral image and signal processing: A comprehensive overview of the state of the art," *IEEE Geoscience and Remote Sensing Magazine*, vol. 5, no. 4, pp. 37–78, 2017.

[3] Jiaojiao Li, Xi Zhao, Yunsong Li, Qian Du, Bobo Xi, and Jing Hu, "Classification of hyperspectral imagery using a new fully convolutional neural network," *IEEE*

Geoscience and Remote Sensing Letters, vol. 15, no. 2, pp. 292–296, 2018.

[4] M.E. Paoletti, J.M. Haut, J. Plaza, and A. Plaza, "A new deep convolutional neural network for fast hyperspectral image classification," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 145, pp. 120 – 147, 2018, Deep Learning RS Data.

[5] Mercedes E Paoletti, Juan Mario Haut, Ruben Fernandez-Beltran, Javier Plaza, Antonio J Plaza, Jun Li, and Filiberto Pla, "Capsule Networks for Hyperspectral Image Classification," *IEEE Transactions on Geoscience and Remote Sensing*, , no. 99, pp. 1–15, 2018.

[6] Ailong Ma, Yanfei Zhong, Da He, and Liangpei Zhang, "Multiobjective subpixel land-cover mapping," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 56, no. 1, pp. 422–435, 2018.

[7] Ruben Fernandez-Beltran, Juan M Haut, Mercedes E Paoletti, Javier Plaza, Antonio Plaza, and Filiberto Pla, "Multimodal probabilistic latent semantic analysis for sentinel-1 and sentinel-2 image fusion," *IEEE Geoscience and Remote Sensing Letters*, vol. 15, no. 9, pp. 1347–1351, 2018.

[8] José M Bioucas-Dias, Antonio Plaza, Nicolas Dobigeon, Mario Parente, Qian Du, Paul Gader, and Jocelyn Chanussot, "Hyperspectral unmixing overview: Geometrical, statistical, and sparse regression-based approaches," *IEEE journal of selected topics in applied earth observations and remote sensing*, vol. 5, no. 2, pp. 354–379, 2012.

[9] Wing-Kin Ma, José M Bioucas-Dias, Tsung-Han Chan, Nicolas Gillis, Paul Gader, Antonio J Plaza, ArulMurugan Ambikapathi, and Chong-Yung Chi, "A signal processing perspective on hyperspectral unmixing: Insights from remote sensing," *IEEE Signal Processing Magazine*, vol. 31, no. 1, pp. 67–81, 2014.

[10] José MP Nascimento and José MB Dias, "Vertex component analysis: A fast algorithm to unmix hyperspectral data," *IEEE transactions on Geoscience and Remote Sensing*, vol. 43, no. 4, pp. 898–910, 2005.

[11] José MP Nascimento and José M Bioucas-Dias, "Hyperspectral unmixing based on mixtures of dirichlet components," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 50, no. 3, pp. 863–878, 2012.

[12] Abderrahim Halimi, Nicolas Dobigeon, Jean-Yves Tourneret, and Paul Honeine, "A new bayesian unmixing algorithm for hyperspectral images mitigating endmember variability," in *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2015, pp. 2469–2473.

[13] Yuki Itoh, Siwei Feng, Marco F Duarte, and Mario Parente, "Semisupervised endmember identification in nonlinear spectral mixtures via semantic representation," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 55, no. 6, pp. 3272–3286, 2017.

[14] Ruben Fernandez-Beltran, Pedro Latorre-Carmona, and Filiberto Pla, "Latent topic-based super-resolution for remote sensing," *Remote Sensing Letters*, vol. 8, no. 6, pp. 498–507, 2017.

[15] Ruben Fernandez-Beltran, Juan M Haut, Mercedes E Paoletti, Javier Plaza, Antonio Plaza, and Filiberto Pla, "Remote sensing image fusion using hierarchical multimodal probabilistic latent semantic analysis," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 11, no. 12, pp. 4982–4993, 2018.

[16] Ruben Fernandez-Beltran and Filiberto Pla, "Sparse multi-modal probabilistic latent semantic analysis for single-image super-resolution," *Signal Processing*, vol. 152, pp. 227–237, 2018.

[17] Ruben Fernandez-Beltran and Filiberto Pla, "Latent topics-based relevance feedback for video retrieval," *Pattern Recognition*, vol. 51, pp. 72–84, 2016.

[18] Ruben Fernandez-Beltran, Antonio Plaza, Javier Plaza, and Filiberto Pla, "Hyperspectral unmixing based on dual-depth sparse probabilistic latent semantic analysis," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 56, no. 11, pp. 6344–6360, 2018.

[19] Raymond Wan, Vo Ngoc Anh, and Hiroshi Mamitsuka, "Efficient probabilistic latent semantic analysis through parallelization," in *AIRS '09 Proceedings of the 5th Asia Information Retrieval Symposium on Information Retrieval Technology*, 2009, pp. 432–443.

[20] David Maxwell Chickering, "Learning bayesian networks is np-complete," in *Learning from data*, pp. 121–130. Springer, 1996.

[21] Ruben Fernandez-Beltran and Filiberto Pla, "Incremental probabilistic latent semantic analysis for video retrieval," *Image and Vision Computing*, vol. 38, pp. 1–12, 2015.

[22] Eli Koffi Kouassi, Toshiyuki Amagasa, and Hiroyuki Kitagawa, "Efficient probabilistic latent semantic indexing using graphics processing unit," *Procedia Computer Science*, vol. 4, pp. 382–391, 2011.

[23] Zhao Liang, Wenye Li, and Yuxi Li, "A parallel probabilistic latent semantic analysis method on mapreduce platform," in *IEEE International Conference on Information and Automation (ICIA)*, 2013, pp. 1–10.

[24] D. M. Blei, "Probabilistic topic models," *Communications ACM*, vol. 55, no. 4, pp. 77–84, 2012.

[25] J. M. Haut, S. Bernabé, M. E. Paoletti, R. Fernandez-Beltran, A. Plaza, and J. Plaza, "Low-high-power consumption architectures for deep-learning models applied to hyperspectral image classification," *IEEE Geoscience and Remote Sensing Letters*, pp. 1–5, 2018.

[26] Mercedes E Paoletti, Juan Mario Haut, Ruben Fernandez-Beltran, Javier Plaza, Antonio J Plaza, and Filiberto Pla, "Deep Pyramidal Residual Networks for Spectral-Spatial Hyperspectral Image Classification," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 57, no. 2, pp. 740–754, 2019.

[27] Delian Liu and Liang Han, "Spectral curve shape matching using derivatives in hyperspectral images," *IEEE Geoscience and Remote Sensing Letters*, vol. 14, no. 4, pp. 504–508, 2017.

[28] Na Li, Xinchen Huang, Huijie Zhao, Xianfei Qiu, Ruonan Geng, Xiuping Jia, and Daming Wang, "Multiparameter optimization for mineral mapping using hyperspectral imagery," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 11, no. 4, pp. 1348–1357, 2018.

[29] Jun Li, Alexander Agathos, Daniela Zaharie, José M Bioucas-Dias, Antonio Plaza, and Xia Li, "Minimum volume simplex analysis: A fast algorithm for linear hyperspectral unmixing," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 53, no. 9, pp. 5067–5082, 2015.

[30] A Huck and M Guillaume, "Robust hyperspectral data unmixing with spatial and spectral regularized nmf," in *2010 2nd Workshop on Hyperspectral Image and Signal Processing: Evolution in Remote Sensing*. IEEE, 2010, pp. 1–4.

[31] Jun Li, José M Bioucas-Dias, Antonio Plaza, and Lin Liu, "Robust collaborative nonnegative matrix factorization for hyperspectral unmixing," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 54, no. 10, pp. 6076–6090, 2016.

[32] Yan Ma, Haiping Wu, Lizhe Wang, Bormin Huang, Rajiv Ranjan, Albert Zomaya, and Wei Jie, "Remote sensing big data computing: Challenges and opportunities," *Future Generation Computer Systems*, vol. 51, pp. 47–60, 2015.

[33] Ruben Fernandez-Beltran and Filiberto Pla, "Prior-based probabilistic latent semantic analysis for multimedia retrieval," *Multimedia Tools and Applications*, vol. 77, no. 13, pp. 16771–16793, 2018.

[34] Michael Theodore Eismann, *Hyperspectral remote sensing*, SPIE Bellingham, 2012.

[35] Feiyun Zhu, Ying Wang, Bin Fan, Shiming Xiang, Geofeng Meng, and Chunhong Pan, "Spectral unmixing via data-guided sparsity," *IEEE Transactions on Image Processing*, vol. 23, no. 12, pp. 5412–5427, 2014.

Análisis y estudio de prestaciones de sistemas de codificación hardware/software para el HEVC: escenario Intra

Rubén Miguélez-Tercero[1], Damián Ruiz-Coll[2], Gerardo Fernández-Escribano[1] y Pedro Cuenca[1]

Resumen — **HEVC (en inglés, High Efficiency Video Coding) es el estándar de compresión de vídeo sucesor del H.264/AVC, el cual alcanza una calidad de video similar a éste con una reducción de la tasa binaria próxima al 50%. Esto es posible gracias a algunas de las novedades que incorpora, como un nuevo particionado de las unidades de codificación y el aumento de modos de predicción Intra-picture a 35 modos, entre otras.**

El objetivo de este artículo es la comparación del rendimiento de diferentes implementaciones de codificadores hardware/software conforme al estándar HEVC, cuyos resultados se establecerán como punto de partida para futuras simulaciones con el futuro estándar bajo desarrollo denominado VCC (en inglés, Versatile Video Coding). La comparativa tendrá en cuenta la medición de diferentes parámetros de eficiencia y complejidad como: la tasa de compresión, la calidad del vídeo mediante una métrica objetiva (PSNR) y el tiempo de codificación. Las diferentes codificaciones se realizarán con unas condiciones fijas, para garantizar que la comparativa es justa.

Para llevar a cabo la comparativa que se persigue en el artículo se han escogido cuatro implementaciones de codificadores HEVC, dos software y dos hardware: el modelo de referencia HM-16.20 y el software libre x265; y dos codificadores hardware específicos de consumo, del fabricante Intel y de Nvidia. Del resultado de la comparativa se concluye que el códec HM-16.20 es el que mejor eficiencia ofrece, sin embargo, el códec basado en arquitectura Intel es el más rápido, pero el que arroja unas prestaciones más modestas en términos de eficiencia de compresión.

Palabras clave —**HEVC, predicción Intra-picture.**

I. Introducción

HEVC (High Efficiency Video Coding) fue aprobado en el mes de enero de 2013 por el grupo de trabajo conocido como Joint Collaborative Team on Video Coding (JCT-VC). Este estándar oficialmente denominado como H.265 por la ITU-R [1] y MPEG-H Parte 2 por el ISO/IEC [2].

Desde que se desarrolló dicho estándar, varios han sido los codificadores tanto hardware como software que se han desarrollado. Siendo el HM el software de referencia desarrollado por el grupo de expertos de la JCT-VC.

El software de referencia de HEVC emplea la fuerza bruta para realizar la codificación Intra, obteniendo un resultado satisfactorio, pero en un tiempo elevado. Este es el principal motivo por el que la comunidad científica y grandes multinacionales (Intel, Nvidia, AMD…) han invertido recursos y tiempo en la elaboración de codificadores HEVC que, dependiendo del escenario, reduzcan la cantidad de información a transmitir o a almacenar.

Este artículo tiene como tarea principal realizar una comparativa de rendimiento entre diferentes códecs de HEVC utilizando exclusivamente el esquema de predicción Intra-picture. El objetivo de ello es poder determinar cuáles tienen un rendimiento óptimo en función de unos determinados parámetros. Se han seleccionado tres parámetros considerados clave en un codificador de vídeo, la tasa de compresión del códec, la calidad objetiva medida por medio de la métrica PSNR (en inglés, Peak Signal-to-Noise Ratio) y el tiempo de codificación. La comparativa se llevará a cabo con una batería de secuencias de vídeo de 8 bits clasificadas en diversas clases propuestas por el grupo de trabajo que desarrolló el estándar HEVC, el JCT-VC [3], y que servirán como punto de partida para un futuro estudio con el estándar de codificación de vídeo bajo desarrollo VVC.

La organización del artículo será la siguiente: la Sección II presenta una breve introducción de la predicción Intra-picture de HEVC, en la Sección III se presentan los distintos códecs con sus características. Seguidamente, se muestran los entornos de realización de los test, los resultados del estudio y su análisis en la Sección IV, mientras que, finalmente la Sección V trata las conclusiones del estudio y el trabajo futuro.

II. La predicción Intra-picture en HEVC

HEVC es el sucesor del estándar H.264/AVC, siendo capaz de conseguir la misma calidad de imagen con una reducción de cerca del 50% de la tasa binaria respecto a su antecesor. Esto es gracias en parte a dos novedades que presenta HEVC, y que tienen una importante implicación en la predicción Intra-picture.

La primera novedad que ha permitido dicha reducción es el particionado de las unidades de codificación, conocidas como CTU (en inglés, Coding Tree Units), que permite un particionado jerárquico en el que cada bloque inicial puede ser iterativamente particionado en bloques de resolución mitad.

La otra novedad que presenta la predicción Intra-picture es su esquema de predicción direccional, llamado *Angular Intra Prediction*, que llega a definir un conjunto de 33 predictores direccionales más 2 no direccionales, modo DC y modo Planar. Estos 35 predictores se pueden aplicar a bloques con una gran variedad de tamaño, desde los más pequeños de 4x4 a los más grandes de 64x64 píxeles. En la Figura 1, se muestran los 33 modos direccionales.

[1] Instituto de Investigación en Informática de Albacete, Universidad de Castilla-La Mancha, Albacete, España.

[2] Universidad Rey Juan Carlos, Fuenlabrada, España.

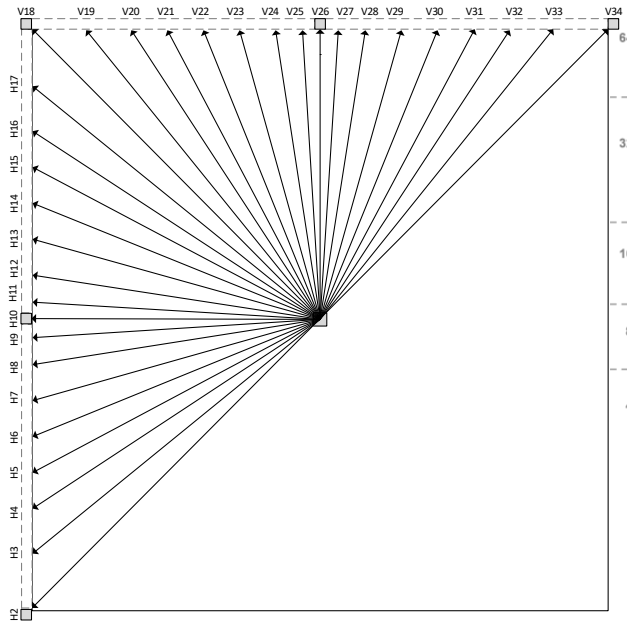


Figura 1. Modos direccionales de predicción Intra de HEVC

En la predicción Intra-picture se omite totalmente la redundancia temporal de los frames que forman parte de una secuencia, realizándose la codificación únicamente con la información del propio frame. En otras palabras, solamente se explota la redundancia espacial. Sin embargo, el problema de ello es el elevado coste computacional que necesita la predicción Intra-picture.

Tal y como se ha indicado previamente, HEVC presenta un esquema de particionado de las unidades de codificación. (CTU). El tamaño máximo de los bloques es de 64x64 píxeles que pueden ser reducidos de forma iterativa mediante la partición en cuatro sub-bloques que presentan la mitad de resolución. Así, hasta alcanzar los bloques de tamaño mínimo que es de 8x8 píxeles.

Una CTU es una colección de varias CUs (en inglés, Coding Units). Una CU define una subpartición de la imagen en regiones cuadradas de tamaño variable. Cada CU puede presentar una o más PUs (en inglés, Prediction Units) y TUs (en inglés, Transform Units). Las distintas CUs pueden ser particionadas en PUs. Las PUs tienen la característica de llegar a alcanzar tamaños mínimos de 4x4 píxeles. En total, pueden presentar cinco tamaños distintos, desde 64x64 píxeles hasta 4x4 píxeles. A su vez, cada CU puede contener una o más TUs, también organizadas en estructura de árbol. Además, es posible que varias TUs estén contenidas dentro de una única PU. Las TUs también tienen un tamaño variable, pudiendo abarcar tamaños desde 32x32 a 4x4 píxeles. En la Figura 2, se muestra un ejemplo de particionado de una CTU en diversas CUs, PUs y TUs de resolución menor.

En el modo de predicción Intra-picture todas las CUs, PUs y TUs tienen un tamaño simétrico, es decir, si el tamaño del nivel superior es 2Nx2N, el tamaño del siguiente nivel es NxN y así sucesivamente. En la Figura 3 se observa el particionado de las CUs de una secuencia de vídeo con el método explicado anteriormente.

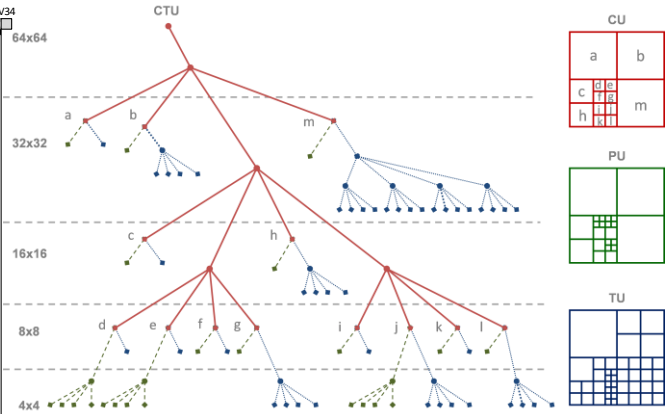


Figura 2. Particionado de una CTU

Figura 3. Particionado de las CUs

Por otro lado, la segunda novedad que presenta HEVC en su modo de codificación Intra-picture son los 35 modos de predicción Intra. A parte del aumento de la cantidad de modos direccionales, HEVC emplea la misma estrategia de predicción intra-picture que fue planteada en H.264/AVC [4][5]. Su estrategia consiste en la explotación de la alta correlación espacial entre un bloque y los píxeles vecinos de ese bloque pertenecientes a los bloques superior e izquierdo. En la Figura 4 se puede observar dicha correlación, siendo los píxeles indicados con una "*V*" los pixeles vecinos utilizados como pixeles de referencia para construir el predictor de cada modo direccional [6]. La misma Figura 4 muestra cómo se utiliza un modo direccional en un sub-bloque de 4x4 píxeles para el modo de predicción horizontal H10, utilizando como predicción los píxeles de referencia de su izquierda. Además, se refleja cómo se tienen en cuenta los píxeles del bloque izquierdo y superior del bloque a codificar [7].

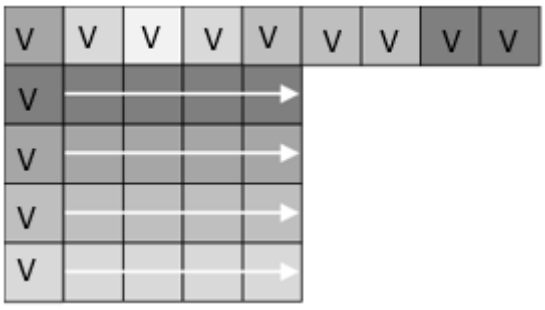


Figura 4. Ejemplo de aplicación de un modo direccional de codificación Intra HEVC

Los 33 modos direccionales cubren los gradientes espaciales en un arco de π radianes, 17 cubren una orientación vertical, definidos de V18 a V34, mientras que los 16 restantes presentan una orientación horizontal y se definen de H2 a H17. Los 33 modos direccionales son eficientes para codificar bloques que tienen un determinado gradiente espacial. Por el contrario, los modos "Planar" y "DC" son más eficientes en bloques con una alta homogeneidad o un ligero gradiente. El codificador escogerá el modo más adecuado para un determinado bloque bajo una cierta función de coste. Uno de estos algoritmos de cálculo de costes es el modelo de búsqueda exhaustiva basado en el RDO (en inglés, Rate Distortion Optimization), aunque hay otros algoritmos que proporcionan un rendimiento aproximado sin ser tan costosos computacionalmente [8].

III. ANÁLISIS DE IMPLEMENTACIONES DEL ESTÁNDAR HEVC

Hoy en día existen multitud de implementaciones del estándar de codificación de vídeo HEVC, y es una tarea costosa y laboriosa hacer una comparativa exhaustiva de todos. Sin embargo, los datos arrojados por estos estudios sirven como base a las implementaciones y mejoras que se realizan sobre un estándar multimedia durante toda la vida útil de éste. En nuestro caso, centramos el estudio en el estándar HEVC, pues como se detallará en uno de los puntos de este artículo, servirá de base y punto de partida para analizar los futuros estándares que están cerca de ver la luz en la industria. Por otro lado, se han escogido para el análisis y estudio cuatro implementaciones más relevantes en la actualidad y que cuentan con implementaciones software y hardware. Las implementaciones seleccionadas son las siguientes:

- HEVC: modelo de referencia del JCT-VC, versión HM-16.20 [9].
- x265: implementación de software libre y código abierto desarrollado inicialmente por VideoLan [10]. También disponible en FFmpeg como códec libx265 [11].
- Intel Quick Sync Video: implementación hardware desarrollada por Intel y disponible en dispositivos con un determinado procesador Intel [12]. Este codificador es accesible desde la librería software incluida en FFmpeg como hevc_qsv [11].
- NVENC: implementación hardware desarrollada por Nvidia y disponible en dispositivos hardware como Jetson Nano y Nvidia GeForce GTX [13]. Este codificador es accesible desde la librería software incluida en FFmpeg como la librería hevc_nvenc [11].

En los siguientes subapartados se procede a detallar las características de cada uno de ellos; los dos primeros son códecs software, mientras que, los dos últimos son dependientes de una arquitectura hardware propietaria.

A. HM-16.20

Esta implementación fue desarrollada por el mismo consorcio de expertos que desarrolló el estándar HEVC, JCT-VC, y constituye el modelo de referencia del estándar HEVC, conocido como HM, el cual ha ido evolucionando hasta su actual versión V.16.20.

El objetivo del HM es proporcionar una base sobre la cual realizar experimentos para determinar qué sistemas de codificación proporcionan mejor rendimiento durante el desarrollo del estándar. Este códec no pretende ser una implementación particularmente eficiente, en términos de coste computacional, por el contrario, trata de explotar casi en su totalidad todas las ventajas que se contemplan en el estándar [9].

Respecto a su funcionamiento, hay que destacar entre la gran cantidad de opciones que soporta, la posibilidad de utilizar el modo de predicción Intra (todas las imágenes de tipo I) u otros modos que combinan distintos tipos de imágenes en función de la configuración del GOP (en inglés, Group Of Pictures) escogido.

Finalmente, este códec emplea un algoritmo de fuerza bruta a la hora de realizar la Intra-prediction, lo que provoca que la codificación requiera de un alto coste computacional, consumiendo unos elevados tiempos de codificación. En el análisis de los resultados de la comparativa se podrá ver con detalle.

B. x265

La implementación x265 de HEVC, está disponible como una librería software *open source* que se publica bajo los términos de la licencia GPL (en inglés, General Public License) de GNU (acrónimo recursivo de "GNU's Not Unix"). Un desarrollador de este códec es VideoLan (fabricante del conocido reproductor VLC Player) que proporciona un repositorio donde están disponible diversas versiones del codificador.

Además, al disponer de una licencia GPL, cualquiera puede realizar modificaciones en el código del software, aunque también está disponible una versión comercial. Eso ha provocado que la versión de x265 disponible en FFmpeg no sea la misma que distribuye VideoLan. Por este motivo, se ha decidido contemplar en el estudio ambas versiones del codificador: la implementación VideoLan [10] y la incluida en FFmpeg como x265 [11]. En la siguiente sección sólo habrá una serie de x265 en las gráficas porque ambas versiones proporcionan las mismas prestaciones en tasa binaria y calidad, variando solo el tiempo de codificación.

Las opciones que este codificador ofrece al usuario son bastantes completas, permitiendo realizar la codificación en el modo Intra-picture con un simple comando. Por otro lado, la salida proporciona una gran cantidad de información: PSNR, bitrate, tamaño fichero de salida, etc.

A la hora de presentar los resultados, la consola no ofrece información del PSNR por lo que es necesario calcularlo con una aplicación externa. Además, el PSNR y el tamaño del fichero codificado es el mismo al tratarse del mismo códec, existiendo únicamente diferencia en el tiempo de compresión de las secuencias, y con independencia de si se ha empleado VLC o FFmpeg.

C. Intel Quick Sync Video

Intel Quick Sync Video es la marca de Intel para identificar su hardware de codificación y decodificación

de video. Quick Sync se introdujo con la microarquitectura Sandy Bridge en enero de 2011, y se ha encontrado en las distintas familias de procesadores de Intel desde entonces. A diferencia de la codificación de video en una CPU (en inglés, Central Processing Unit) o una GPU (en inglés, Graphics Processing Unit) de propósito general, Quick Sync es un núcleo de hardware dedicado en el procesador [12].

Este códec puede ser accedido desde el software FFmpeg [11] para su ejecución, y se utilizará dicho software para codificar las secuencias con 2 procesadores Intel diferentes, un i7 de la microarquitectura SkyLake (6º generación de Intel) y otro i7 de la microarquitectura Coffee Lake (8º generación).

A diferencia de los códec HM-16.20 y del x265, no ofrece un comando específico para indicar que se emplee el modo Intra-picture por lo que se utilizará una configuración con tamaño de GOP igual a 1.

Por otro lado, y como veremos en el apartado de evaluación de prestaciones, solamente aparecerá una serie correspondiente a este procesador, a pesar de que se haya realizado dos simulaciones con distinta microarquitectura. El motivo de ello es que, al tratarse del mismo códec el resultado es el mismo en términos de PSNR y tasa binaria, variando únicamente el tiempo de ejecución, debido a que el procesador Intel Coffe Lake trabaja con una frecuencia de reloj más elevada, pero ejecuta los mismos algoritmos.

D. NVENC

A partir de las GPU de Nvidia de la generación Kepler, todas las tarjetas gráficas de este fabricante implementan un codificador NVENC, que proporciona una codificación de vídeo basada en hardware totalmente acelerada. Es capaz de codificar vídeos con una resolución de hasta 8K, aunque depende de la GPU empleada [13].

NVENC se puede acceder desde el software FFmpeg. Para poder realizar las ejecuciones sobre el modo Intra-picture hay que seleccionar el tamaño de GOP a 1 y no se incluye información de salida acerca del PSNR, por lo que esta información se obtiene con una aplicación externa al códec.

La obtención de resultados con este codificador se ha llevado a cabo con una GPU Nvidia GeForce GTX 1080TI.

IV. ANÁLISIS Y ESTUDIO DE CODIFICADORES HEVC

Para hacer una comparativa con unas condiciones de simulación similares para todos los codificadores, atendiendo a ciertas particularidades que estos presentan, se ha elegido una batería de 22 secuencias de vídeo de 8 bits elaboradas por la JCT-VC. La ejecución de las pruebas para codificar estas secuencias se ha llevado a cabo con las condiciones de codificación establecida por el consorcio para el perfil “Main-Intra”, denominadas como CTC (en inglés, Common Test Conditions). [3].

Este conjunto de secuencias de test cubre un extenso rango de resoluciones. La mayor resolución es de Ultra-HD (realmente están recortadas de UHD) y la menor de 416x240 píxeles, destacando una importante variedad de complejidades espacio-temporales entre las secuencias. Dichas secuencias se agrupan en seis clases en función de sus características: su resolución y la naturaleza. Las primeras cinco clases se caracterizan por su contenido natural, y la última es de naturaleza sintética, ya que sus contenidos proceden de videojuegos y de capturas de ordenador. Las resoluciones de las secuencias por clase son las siguientes: 2500x1600 (clase A), 1920x1080 (clase B), 832x480 (clase C), 416x240 (clase D) y 1280x720 (clase E). La clase F presenta secuencias con distintas resoluciones (832x480, 1024x768 y 1280x720).

Con el objetivo de llevar a cabo una comparativa neutral entre los diferentes códecs, se ha tenido que emplear las condiciones más similares posibles en los diferentes ordenadores utilizados. En primer lugar, al pretender realizar una comparativa del rendimiento del códec de Intel entre dos procesadores distintos, ha implicado utilizar dos ordenadores para el códec de Intel. A parte de ello, se ha empleado un tercer ordenador con una tarjeta gráfica de Nvidia compatible con el códec de Nvidia.

A continuación, en la Tabla I, se indica las características de los ordenadores en los que se han ejecutado las pruebas de los códecs.

Tabla I. Características de los ordenadores de pruebas

	PC 1	PC2	PC 3
Sistema operativo	Windows 10	Windows 10	Windows 10
Memoria RAM	8 GB	8 GB	16 GB
CPU	Intel i5 3GHz	Intel i7 3,2 GHz	Intel i7 3,4 GHz
GPU	Nvidia GeForce GTX 1080TI	Intel UHD Graphics 360	Intel UHD Graphics 360

En el PC1 se han ejecutado las pruebas correspondientes al códec de Nvidia, las 2 pruebas del códec x265 (FFmpeg y VideoLan) y la del codificador de referencia HM-16.20. Por otro lado, en el PC2 se ha realizado la prueba del códec de Intel con el procesador de la familia Coffee Lake. Por último, en el PC3 se ha ejecutado la prueba restante del códec de Intel con el procesador de la familia SkyLake.

Tal y como se observa, las características de las tres máquinas son prácticamente similares como para considerar que el entorno de pruebas es idéntico para todos los codificadores, ya que las diferencias que existen no radican de manera considerable en el funcionamiento de los códecs. Además, todo el peso de las pruebas con el códec de Nvidia se realizan sobre la arquitectura de la tarjeta gráfica.

A. Resultados de eficiencia de compresión (calidad-tasa binaria)

La información que se pretende analizar y comparar es la calidad objetiva del vídeo codificado, que se obtiene por medio de la métrica PSNR que se define con la ecuación 1, junto con el bitrate de la secuencia de salida. Se recogerá esa información para 4 valores diferentes de QP (en inglés, Quantization Parameter), idénticos valores de QP para cada secuencia. Los valores de QP son los siguientes: 22, 27, 32 y 37 [3]. A menor factor de

cuantificación, el bitrate será mayor, y en consecuencia la calidad obtenida también será mayor (PSNR). Conforme aumenta el QP, esos dos parámetros se reducen [14].

$$PSNR = 10 \times log_{10} \frac{255^2}{MSE} \quad (1)$$

Con ello, se obtiene una gráfica por cada una de las 22 secuencias, en la que se observa la eficiencia de cada códec en función de la calidad y bitrate. Estas gráficas se conocen como curvas RD (Rate/Distorsion). El códec de más calidad es el que obtenga un PSNR más elevado con una menor tasa binaria, es decir, cuya curva esté por encima del resto [14].

Como hay 22 secuencias, en este artículo se mostrarán únicamente 3 gráficas correspondientes a secuencias de distintas clases, pues se ha considerado que son las más destacables y representativas; representan de manera fiel y aproximada un resumen del resto. En la Figura 5 se muestra la gráfica de la secuencia "*Johnny*" de clase E, la Figura 6 se corresponde a la secuencia "*BasketballDrillText*" de clase F y finalmente, la Figura 7 representa la gráfica de la secuencia "*BQSquare*" de clase D.

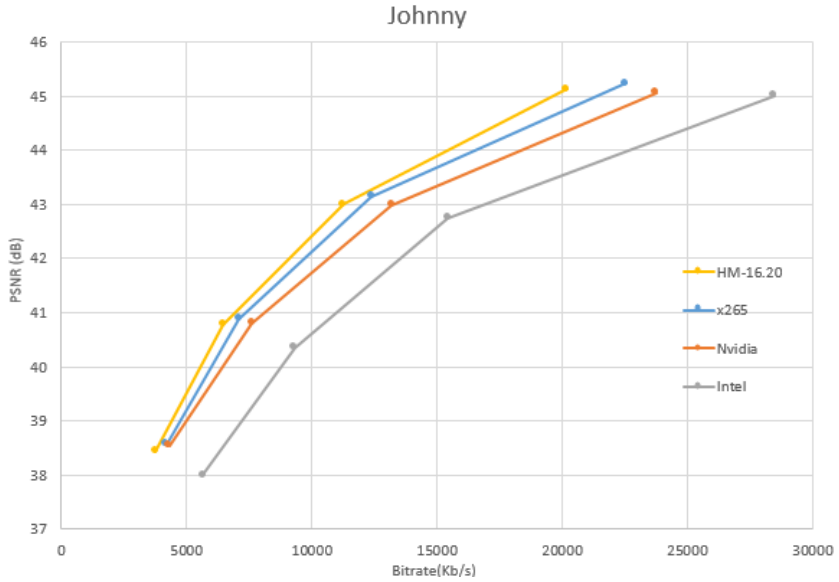


Figura 5. Curva RD secuencia "*Johnny*" (Clase E)

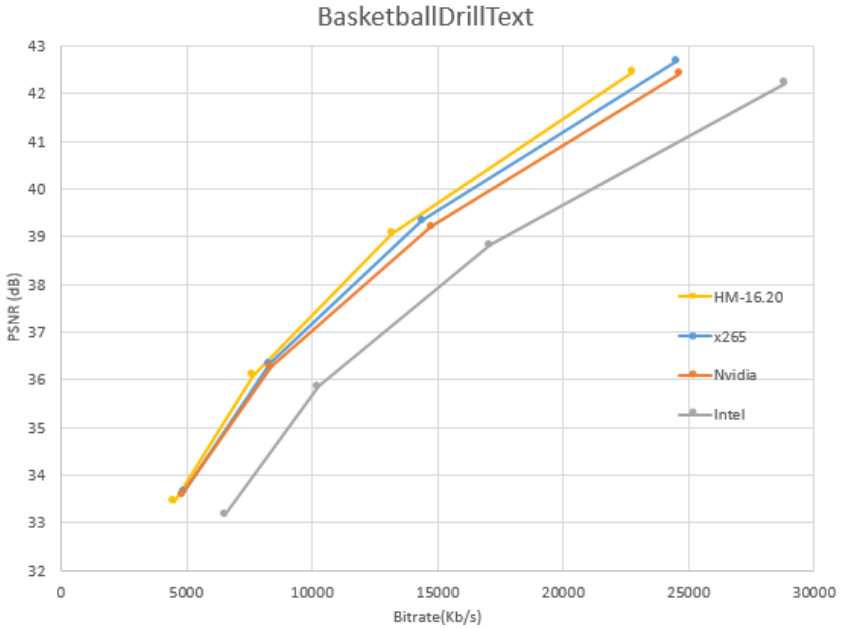


Figura 6. Curva RD secuencia "*BasketballDrillText*" (Clase F)

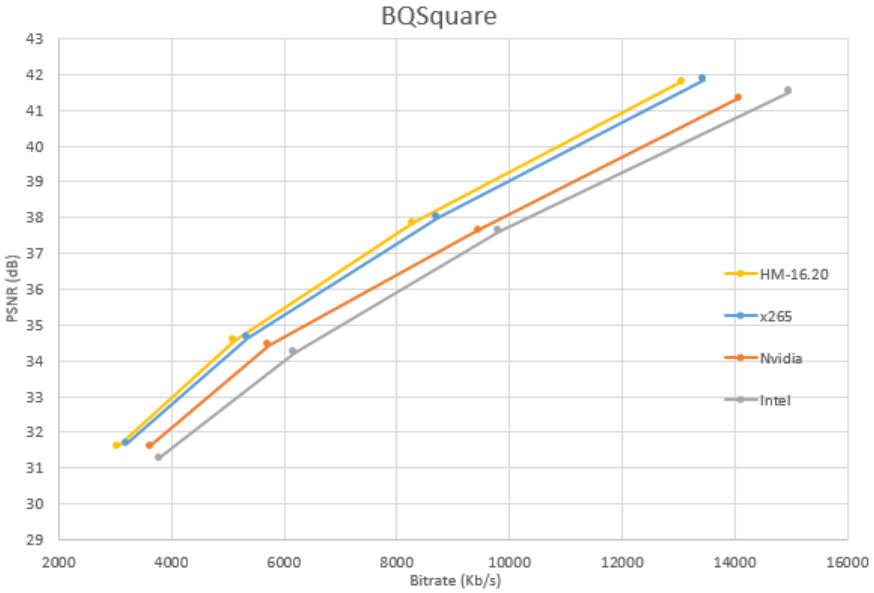


Figura 7. Curva RD secuencia "BQSquare" (Clase D)

En las tres gráficas se observa claramente que el modelo de referencia, el códec HM-16.20, es el que obtiene un rendimiento superior al resto en términos de eficiencia de compresión. El motivo de ello es que emplea un algoritmo de alto coste computacional para la Intra-picture, así como para otras características que se soportan en este estándar, principalmente características basadas en la fuerza bruta y en las que se prueban todas las combinaciones posibles, seleccionando la más eficiente. La desventaja de ello es el alto tiempo de cómputo que analizaremos en la Tabla II y en la Tabla III.

El codificador de Intel es el que presenta un menor rendimiento en términos de eficiencia de compresión a la vista de los resultados, exceptuando en la secuencia de BQSquare (Figura 7), en la que su rendimiento está más próximo al resto.

Respecto al codificador desarrollado por Nvidia y al x265, ambos tienen un rendimiento muy parejo y próximo al HM-16.20, especialmente el x265. En la gráfica de la Figura 6, que se corresponde a una secuencia de origen natural, para un bitrate bajo (QP alto) el rendimiento del Nvidia y x265 es prácticamente el mismo que el del HM-16.20. Sin embargo, conforme el QP se reduce y se aumenta el bitrate, la situación cambia y el HM-16.20 presenta mayor diferencia de prestaciones.

Por el contrario, en la gráfica de la Figura 7, los códecs software presentan una mayor diferencia de rendimiento que los hardware, teniendo el x265 unas prestaciones muy próximas al HM-16.20. En definitiva, estas tres gráficas representan un resumen del comportamiento de los cuatro códecs con el conjunto de secuencias de test teniendo en cuenta la calidad y la tasa binaria.

B. Resultados de tiempo de codificación

Hay que indicar que, existe otro factor importante que no se ha tenido en cuenta hasta este momento del análisis, y que es el tiempo de cómputo o también conocido como tiempo de codificación. En la Tabla II y en la Tabla III, se pueden observar los tiempos de codificación en frames por segundo o fps, empleados por cada implementación en las distintas secuencias del conjunto de test. En ambas tablas, las secuencias se encuentran agrupadas en clases y al final de cada tabla se muestra el tiempo medio de compresión de las secuencias por clases y el tiempo total medio del conjunto de secuencias. Los tiempos se presentan en fps (frames por segundos). La Tabla II agrupa los codificadores de implementación software y la Tabla III los de implementación hardware.

Tabla II. Tiempo compresión secuencias test implementaciones software (fps)

Secuencia	HM-16.20	x265 VideoLan	x265 FFmpeg
Traffic	0,05	0,18	2,02
PeopleOnStreet	0,05	0,17	2,07
BasketballDrive	0,10	0,39	4,45
BQTerrace	0,08	0,36	3,81
Cactus	0,09	0,36	3,90
Kimono	0,10	0,40	4,53
ParkScene	0,09	0,35	3,79
BasketballDrill	0,45	1,79	19,25
BQMall	0,45	1,86	19,78
PartyScene	0,35	1,49	14,78
RaceHorses	0,43	1,73	18,06
BasketballPass	1,83	6,66	69,06
BQSquare	1,47	5,73	54,68
BlowingBubbles	1,44	5,41	51,19
RaceHorses	1,67	6,02	59,65
FourPeople	0,22	0,92	10,54
Johnny	0,24	0,99	11,70
KristenAndSara	0,24	0,97	11,25
BasketballDrillText	0,43	1,78	18,99
ChinaSpeed	0,23	0,96	10,50
SlideEditing	0,17	0,82	8,33
SlideShow	0,26	1,20	14,22
Clase A	0,05	0,18	2,07
Clase B	0,09	0,37	4,10
Clase C	0,42	1,72	17,97
Clase D	1,60	5,95	58,64
Clase E	0,24	0,96	11,16
Clase F	0,27	1,19	13,01
Total	0,47	1,84	18,93

Tabla III. Tiempo compresión secuencias test implementaciones hardware (fps)

Secuencia	Nvidia	Intel Coffee Lake	Intel SkyLake
Traffic	36,37	94,93	67,81
PeopleOnStreet	35,83	88,13	61,84
BasketballDrive	65,87	226,80	166,25
BQTerrace	56,32	207,40	149,13
Cactus	72,95	214,29	154,01
Kimono	71,70	209,89	150,27
ParkScene	70,17	207,59	149,17
BasketballDrill	301,32	551,94	362,62
BQMall	311,50	576,40	376,15
PartyScene	294,52	545,10	355,33
RaceHorses	271,51	557,29	371,53
BasketballPass	762,10	1.415,22	833,45
BQSquare	884,66	1.449,82	830,31
BlowingBubbles	726,42	1.341,10	780,69
RaceHorses	629,56	1.181,95	572,31
FourPeople	58,44	408,74	281,65
Johnny	44,51	450,96	311,69
KristenAndSara	45,34	434,16	305,01
BasketballDrillText	298,02	548,81	309,73
ChinaSpeed	56,21	329,62	222,97
SlideEditing	139,22	316,73	214,12
SlideShow	55,03	473,93	315,57
Clase A	36,10	91,53	64,82
Clase B	67,40	213,19	153,77
Clase C	294,71	557,68	366,41
Clase D	750,69	1.347,02	754,19
Clase E	49,43	431,29	299,45
Clase F	137,12	417,27	265,60
Total	240,34	537,76	333,71

Además, en la Tabla IV, se muestra el tiempo total de compresión de cada codificador en horas. Con bastante diferencia y como era de esperar, el códec HM-16.20, es el codificador que tarda más tiempo en realizar todas las compresiones. Es de destacar que, el más rápido es el codificador de Intel seguido de cerca por el de Nvidia. En tercera posición se ubica el códec x265.

Tabla IV. Tiempo total de codificación de las 22 secuencias test por codificador

Codificador	**Tiempo total**
HM-16.20	56,88 horas
x265 VideoLan	13,82 horas
x265 FFmpeg	1,28 horas
Nvidia	0,14 horas
Intel SkyLake	0,11 horas
Intel Coffee Lake	0,03 horas

Respecto al codificador de Intel, las pruebas realizadas arrojan que las codificaciones sobre una CPU de 8º generación, como son las CPUs de la familia Coffee Lake de altas prestaciones, conlleva un gran ahorro de tiempo, en relación con otro procesador de una generación anterior. En este caso, la comparación se ha hecho con un CPU de la familia SkyLake (6º generación), comprobando que se puede reducir el tiempo de codificación en torno a 73%.

Sobre el códec x265, se observa cómo el tiempo de codificación varía entre ambas versiones. La versión de VideoLan es más antigua y de código abierto, mientras que, la versión que se encuentra integrada en FFmpeg es fruto de la evolución de la versión original de VideoLan, presentando un ahorro de tiempo significativo. El ahorro de tiempo de codificación de la versión de FFmpeg respecto a la de VideoLan es de un 90,74%.

V. Conclusiones y trabajo futuro

Del análisis del estudio comparativo llevado a cabo en la sección anterior se obtienen una serie de conclusiones que se irán desgranando a continuación.

En primer lugar, analizando los datos de las curvas RD y los tiempos de codificación, no existe el codificador perfecto, es decir, un codificador que supere al resto tanto en eficiencia de compresión como en tiempo de compresión. Los cuatro codificadores analizados destacan en alguna de las características evaluadas, eficiencia de compresión o tiempo de codificación, manteniendo un buen balance. Por ejemplo, el HM-16.20 destaca al ser la implementación más eficiente en términos de compresión, y el códec de Intel y el de Nvidia destacan por su capacidad de cómputo, con unos tiempos de codificación muy bajos. Por el contrario, el x265 presenta buen balance entre calidad y velocidad de procesado.

Este hecho implica que en función de las necesidades y/o características de la secuencia haya que emplear un codificador u otro. En el caso de que se pretenda llevar a cabo una codificación en tiempo real, esta no se podría llevar a cabo con los codecs software, debido a su bajo rendimiento computacional. Obviamente el más indicado sería el códec de Intel o bien el de Nvidia. En el caso de una película que se quiera grabar en la mejor calidad posible es recomendable emplear un códec con una prestación alta en calidad como es el HM-16.20. Además, siempre está presente el caso intermedio, el cual no se tiene como objetivo conseguir la mayor calidad posible y no se dispone de un ancho margen de tiempo para codificar. En esa ocasión un códec aceptable es el x265.

En definitiva, es beneficioso para el mercado y para los consumidores la existencia de codificadores con características que permitan adaptarse a distintas necesidades, calidad o tiempo de codificación.

Sobre el codificador de x265, se recomienda a los usuarios que utilicen la versión que viene incluida en FFmpeg. Los motivos son obvios, presenta un ahorro del 90,74% en tiempo de compresión respecto a la versión de VideoLan.

Por último, el trabajo llevado a cabo en este artículo presenta varias líneas de trabajo futuro. Se pretende trabajar con sistemas embebidos para analizar su eficiencia y prestaciones de compresión. Para ello, se van a realizar pruebas en Raspberry Pi (con el estándar H.264, ya que no soporta HEVC), en la Nvidia Jetson Nano [15] y en la Xilinx [16], en HEVC. Se considera interesante este estudio para analizar la diferencia de rendimiento entre las máquinas empleadas en este estudio y los sistemas embebidos, para conocer la diferencia de rendimiento entre los propios sistemas embebidos.

También es importante tener en cuenta el elevado uso de los dispositivos móviles a día de hoy. Uno de los inconvenientes de estos dispositivos es la duración de la batería, que normalmente no es la deseada por

los usuarios, por lo que los fabricantes de estos dispositivos móviles deberán optimizar sus dispositivos para permitir unas prestaciones de vídeo adecuadas, sin que suponga un elevado consumo de batería. De esta forma, al estudio llevado a cabo en este artículo se propone añadir el estudio de consumo de energía por parte de los diferentes codificadores.

Otra línea de trabajo futuro es comparar las prestaciones del estándar HEVC con el que será su sucesor, el VVC que se promete un 40% más de eficiencia. El VVC ha sido desarrollado por el mismo grupo de trabajo que el HEVC. También se plantea la comparativa con el estándar AOMedia Video 1 (AV1) y que se encuentra libre de licencias por su uso [17].

AGRADECIMIENTOS

El presente trabajo ha sido financiado mediante la Junta de Comunidades de Castilla-La Mancha bajo el proyecto de referencia SBPLY/17/180501/000353, por el Ministerio de Ciencia, Innovación y Universidades del Gobierno de España bajo el proyecto de referencia RTI2018-098156-B-C52, y por la ayuda dirigida a grupos en el marco del Plan Propio de Investigación de la Universidad de Castilla-La Mancha con referencia 2019-GRIN-27060, todas ellos susceptibles de cofinanciación por el Fondo Europeo de Desarrollo Regional (FEDER) .

REFERENCIAS

[1] Rec. ITU-T H.265 and ISO/IEC 23008-2, High Efficiency Video Coding, 2013.

[2] «ISO/IEC 23008-2:2013 Information technology -- High efficiency coding and media delivery in heterogeneous environments -- Part 2: High efficiency video coding».

[3] F. Bossen, «Common test conditions and software reference configurations,» Génova, 2013.

[4] I. E. Richardson, The H.264 Advanced Video Compression Standard, John Wiley & Sons, 2010.

[5] ITU-T, «Recommendation ITU-T H.264,» Génova, 2017.

[6] V. Sze, M. Budagavi y G. J. Sullivan, «High Efficiency Video Coding (HEVC). Algorithms and Architectures,» Springer, Dallas, 2014.

[7] E. Georgiana Paraschiv, D. Ruiz-Coll y G. Fernández-Escribano, «Predicción Intra-picture: algoritmo MDV-SW vs H.264/AVC,» Albacete, 2017.

[8] D. Ruiz-Coll, J. L. Martínez, G. Fernández-Escribano y P. Cuenca, «Análisis estadístico de la eficiencia de la predicción direccional intra-cuadro en HEVC,» Albacete, 2015.

[9] F. Bossen, D. Flynn, K. Sharman y K. Sühring, «HM Software Manual,» 2014.

[10] VideoLan, «x265,» 2018. [En línea]. Available: https://www.videolan.org/developers/x265.html.

[11] FFmpeg, «FFmpeg,» 2019. [En línea]. Available: https://ffmpeg.org/.

[12] Intel, «Intel® Quick Sync Video,» 2018. [En línea]. Available: https://www.intel.co.uk/content/www/uk/en/architecture-and-technology/quick-sync-video/quick-sync-video-general.html?_ga=2.210245973.1114512435.1557316770-2065669479.1552319480.

[13] Nvidia, «NVIDIA VIDEO CODEC SDK,» 2018. [En línea]. Available: https://developer.nvidia.com/nvidia-video-codec-sdk.

[14] G. J. Sullivan, J.-R. Ohm, W.-J. Han y T. Wiegand, «Overview of the High Efficiency Video Coding (HEVC) Standard,» Génova, 2012.

[15] Nvidia, «Nvidia Jetson Nano,» 2019. [En línea]. Available: https://www.nvidia.com/es-es/autonomous-machines/embedded-systems/jetson-nano/.

[16] Xilinx, «Xilinx,» 2019. [En línea]. Available: https://www.xilinx.com/.

[17] C. Feldmann, «Multi-Codec DASH Dataset: An Evaluation of AV1, AVC, HEVC and VP9,» 2018.

Caracterización vial en base a nubes de puntos LiDAR terrestre con MPI

A. M. Esmorís, J. C. Cabaleiro, D. L. Vilariño y F. F. Rivera [1]

Resumen— **Este trabajo se centra en la implementación paralela de un algoritmo aplicable a nubes de puntos LiDAR terrestre, que corresponde a la primera etapa de un conjunto de soluciones dedicadas la detección de bordillos. Se proponen distintas alternativas para abordar el problema de una carga de trabajo desbalanceada y se analiza el impacto de la configuración de uso de la infraestructura computacional del CESGA en el rendimiento. Tras analizar los resultados se ha visto que la estrategia de balanceo dinámico de la carga propuesta ha dado mejores resultados que las dos estrategias de balanceo estático probadas. Finalmente, se ha observado que usar una configuración de ejecución basada en reducir la congestión del bus de memoria contribuye a mejorar significativamente el rendimiento.**

Palabras clave— **MPI, LiDAR, Paralelismo, Computación Distribuida, HPC**

I. Introducción

La tecnología LiDAR terrestre [9][10] aplicada a entornos urbanos, específicamente a la segmentación, clasificación y análisis de elementos viales, supone el procesamiento de grandes volúmenes de datos. Concretamente, estos datos se han obtenido con tecnología LiDAR terrestre móvil [11]. Las nubes de puntos han sido generadas mediante un vehículo dotado con escáneres LiDAR [12] que recorre las zonas deseadas. De manera más específica, por cada metro cuadrado pueden tenerse cientos o miles de puntos distribuidos de modo no uniforme, lo que da lugar a etapas de procesamiento que necesitan un tiempo relevante y limitante en muchas aplicaciones para completarse.

En este trabajo se presenta la paralelización [2][3] de una etapa de procesamiento a la que se le ha llamado *Algoritmo de adyacencia*, que supone el coste computacional más elevado de entre todas las operaciones necesarias. Se utiliza MPI [4][5], una librería para el paso de mensajes en entornos de computación de altas prestaciones. Además, se usa el supercomputador FinisTerrae-II del Centro de Supercomputación de Galicia (CESGA) [1] y se analiza el rendimiento con distintas configuraciones.

Para analizar el rendimiento, se ha tomado como referencia una nube de puntos que corresponde a una calle de la ciudad de A Coruña, con aproximadamente 15 millones de puntos. De entre todas las operaciones que se realizan sobre esta nube, la que más tiempo necesita es la aplicación del algoritmo de adyacencia, que supone casi un 50 % del tiempo de ejecución total. Esto puede comprobarse en los datos presentados en la tabla I, en la cual se expone el conjunto de todas las operaciones que realiza el clasificador de elementos viales para generar el resultado final. En dicha tabla se exponen las etapas de procesamiento cuya ejecución tarda más tiempo, agrupando las etapas menos costosas en la categoría de otros procesamientos por un lado y las operaciones relativas a entrada/salida de datos por el otro.

TABLA I
Tabla de tiempos de ejecución secuencial del clasificador vial

OPERACIÓN	TIEMPO (s)
Alg. de adyacencia	685.89
Alg. de contagio	183.80
Obtener línea virtual	379.58
Proporcionalidad por puntos	49.09
Reconstruir franjas	24.53
Otros procesamientos	4.52
Entrada/Salida	74.93
Total	1401.76

II. Algoritmo de adyacencia

El algoritmo de adyacencia recorre de manera iterativa la nube de puntos, procesando aquellos que no estén previamente etiquetados ni como puntos de carretera ni como puntos de señal horizontal. Por cada uno de estos puntos se determinan sus vecinos en un radio determinado y se considera el punto de carretera más cercano como el CRP (*Closest Road Point*) del punto que se está procesando. A continuación, se aplican dos etapas de procesamiento, salvo que para alguna de ellas se haya especificado un valor de entrada nulo:

1. Procesamiento de obstáculo vertical. En aquellos casos en que exista un punto vecino al que se está analizando, cuya altura supere el umbral especificado, no se considerará que el punto que se está procesando corresponda a un bordillo.
2. Procesamiento de diferencia de altura. Cuando la diferencia de altura entre el punto que se está analizando y su CRP supere un máximo indicado como argumento, no se considerará el punto que se está procesando como punto de bordillo.

Es importante tener en cuenta que el hecho de que un punto no se considere como punto de bordillo tras este procesamiento no implica que no vaya a ser clasificado como tal en etapas posteriores.

El comportamiento del algoritmo de adyacencia se describe en el pseudocódigo del algoritmo 1.

[1]Centro Singular de Investigación en Tecnoloxías Intelixentes, CiTIUS, USC

Algorithm 1 Algoritmo de adyacencia

```
for all p in points do
  compute(p)
end for
function compute(p)
    neighs ← points in radius r from p
    ra ← false // ra := road adjacency flag
    aa ← false // aa := above adjacency flag
    for all q in neighs do
      if q is road then
            ra ← true
            if q is closest to p than crp then
                  crp ← q
            end if
            if q.z − p.z > th1 then
                  aa ← true
            end if
      end if
    end for
    hdiff ← p.z − crp.z
    if ra and not aa and hdiff > 0 then
      if th2 is null or hdiff ≤ th2 then
            p.curb ← true
      end if
    end if
end function
```

III. Balanceo estático de la carga de trabajo

Se ha observado que en la versión paralela el desbalanceo de la carga es muy alto, ya que el algoritmo de adyacencia presenta un lazo cuyo coste depende del tipo y número de vecinos de cada punto y ese valor cambia mucho en las nubes de puntos valoradas. A continuación, se introducen las técnicas utilizadas para aliviar este problema de rendimiento.

A. Distribución por bloques

La primera estrategia consiste en distribuir la carga de trabajo asignando bloques de puntos consecutivos a cada procesador. En primer lugar, se recorre la nube de puntos para determinar cuáles han de ser procesados, estableciendo que no se deben considerar los puntos de carretera ni de señal horizontal como candidatos a bordillo y que, por tanto, no supondrán carga computacional. Una vez conocido el total de puntos que se debe procesar, se dividen en bloques de puntos consecutivos de igual tamaño y se reparten equitativamente entre todos los procesos.

B. Distribución cíclica

Atendiendo a la distribución de las nubes de puntos, puede observarse una tendencia a que los puntos próximos en el archivo de datos estén distribuidos en un conjunto de curvas irregulares, tal y como se ilustra en la figura 1. Generalmente en estas curvas, cuanto más central sea el punto respecto de la misma, mayor será el número de vecinos en un radio reducido y, en consecuencia, mayor será la carga computacional derivada de su procesamiento.

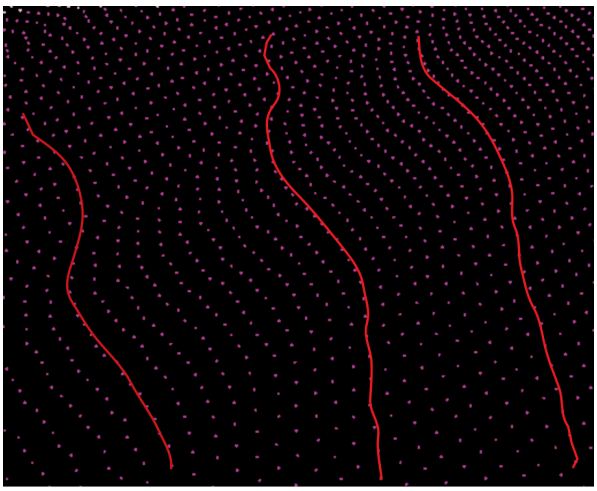


Fig. 1
Distribución de puntos LiDAR terrestre

Esto no siempre se cumple, ya que puntos próximos en el espacio no tienen por qué estar necesariamente próximos en el archivo de datos. Aspectos como pertenecer a distintos barridos, o el procesamiento de los datos originales por un software de terceros para obtener la nube de puntos, explican este hecho.

La falta de relación directa entre la proximidad espacial geográfica y la proximidad en el archivo de datos puede verse más claramente en las nubes de puntos que presentan rotondas. Pudiendo observarse, también, cuando el vehículo que porta el sensor hace un recorrido en el que varía la dirección significativamente, de manera que vuelva a obtener datos de zonas por las que ya había pasado previamente.

Pese a que la distribución de los puntos vecinos expuesta no es siempre uniforme, se considera que es lo suficientemente frecuente como para favorecer distribuciones más balanceadas de la carga de trabajo basándose en ella. Por consiguiente, se propone realizar una distribución basada en un esquema cíclico como segunda alternativa.

IV. Carga de trabajo dinámica temporizada

Incluso realizando una distribución cíclica, que tiende a un reparto más uniforme de la carga de trabajo que la distribución por bloques, ambos métodos dan lugar a una asignación de carga de trabajo poco equilibrada. Este desbalanceo queda patente en la tabla II, donde se muestra para distinto número de procesos el tiempo que tardaron el primer y último proceso en terminar. Dicho desbalanceo redunda directamente en una escasa escalabilidad.

Con el fin de ajustar mejor la distribución de la carga de trabajo, se ha diseñado e implementado un algoritmo de balanceo de carga dinámico que trata de solapar computacion y comunicaciones [6] basándose en el uso de funciones MPI no bloqueantes [7][8]. Dicho algoritmo se divide en dos partes, el sistema de balanceo de carga y el esquema de comunicaciones.

TABLA II
TABLA DE TIEMPO MÁXIMO Y MÍNIMO

N-Procesos	T min	T max	TΔ
4	288.98s	304.38s	15.4s
6	221.8s	256.06s	34.26s
8	193.3s	239.9s	46.6s
12	166.8s	185.9s	19.1s
16	141.3s	173.8s	32.5s
18	128.5s	171.6s	43.1s
24	125.79s	190.8s	65.01s

A. *Sistema de balanceo de carga*

El sistema de balanceo de carga propuesto se basa en poder monitorizar de manera eficiente, en un instante dado, la carga de trabajo completada o *CWL* (*Completed Workload*) por cada proceso. Esta medida debe estar definida en el intervalo [0,1], como la fracción del trabajo total ya realizado.

Sea W el vector que contiene todos los procesos, se ordena de mayor a menor CWL mediante el algoritmo Quicksort [13] y se divide en dos vectores, siendo C el vector que contiene la primera mitad y S el vector que contiene la segunda mitad. En caso de haber un número impar de procesos, siempre será el vector C el que tenga un proceso más que el vector S. De esta manera, el proceso con mayor carga de trabajo completada será el primer elemento en el vector C y el proceso con la menor carga de trabajo completada será el último elemento en el vector S.

Además de la métrica de carga de trabajo completada, el algoritmo debe ser configurado mediante dos parámetros adicionales:

1. WR_{th}. Sirve como umbral de carga de trabajo y, debe haber al menos un proceso que haya completado esta cantidad de carga de trabajo antes de que tenga lugar ninguna redistribución. Ningún proceso que no haya alcanzado este umbral aceptará recibir trabajo de otro proceso. Así, se puede controlar el comienzo de las redistribuciones y favorecer que haya pocas comunicaciones de gran tamaño, que suelen ser preferibles –por el overhead intrínseco a cada comunicación– a muchas comunicaciones de tamaño reducido.
2. WR_{diff}. Sólo se intercambiará trabajo entre pares de procesos cuya diferencia de carga de trabajo supere el valor de este parámetro.

El proceso de redistribución del trabajo (R), asumiendo n procesos y suponiendo que se supera el umbral WR_{th}, es de la siguiente manera:

$$\forall\, i\ c_i \in C\,,\ s_i \in S$$

$$\forall\, 0 \leq i < \left\lceil \frac{n}{2} \right\rceil\ [CWL_i \geq (CWL_{n-i-1} + WR_{diff})]$$

$$\Rightarrow\ R(C_i, S_{\lfloor \frac{n}{2} \rfloor - i - 1})$$

En cuanto a la cantidad de trabajo redistribuido (RWL) desde la perspectiva de S, viene determinado por la siguiente expresión:

$$c \in C\,,\ s \in S$$

$$RWL(c,s) =$$

$$min[0.5, (CWL_c - CWL_s) \cdot (1 - CWL_s)]$$

En el segundo parámetro del mínimo de la anterior formulación, el primer factor corresponde a la diferencia de carga de trabajo entre c y s. Por otro lado, el segundo factor hace alusión a la carga de trabajo pendiente de s. Además, cabe tener en cuenta que existe una restricción técnica impuesta por las comunicaciones, dado que la carga de trabajo nunca podrá superar el tamaño prefijado del buffer de comunicaciones. También, cuando un proceso haya completado toda su carga de trabajo, la proporción de trabajo que reciba de s será siempre de la mitad, es decir, 0.5. Por último, tras la redistribución, las cargas de trabajo del par de procesos implicados quedarán tal que:

$$R_t(c,s) \Rightarrow \begin{cases} CWL_{c_t} < CWL_{c_{t-1}} \\ CWL_{s_t} = CWL_{s_{t-1}} + RWL(c,s) \end{cases}$$

Cabe mencionar que, la carga de trabajo de c tras la redistribución no se puede precisar con exactitud. Esto se debe a que el número de iteraciones computadas por c es desconocido desde la perspectiva de s. Además, la nueva carga no es deducible a partir del CWL de c, puesto que éste puede haber participado ya en otros procesos de redistribución. En consecuencia, no es posible conocer, desde la perspectiva de s, la proporción de carga de trabajo que supone para c la cantidad de trabajo redistribuida.

En la figura 2 se ilustra, mediante un ejemplo, el proceso de redistribución de carga de trabajo con una configuración tal que $WR_{diff} = 0.05$, donde cada procesador trabaja a un ritmo diferente (P0 y P4 en intervalos de 0.2, P1 y P3 en intervalos de 0.1, P2 en intervalos de 0.05 y P5 en intervalos de 0.15). De no realizar operaciones de balanceo de carga, se producirá una pérdida importante de rendimiento.

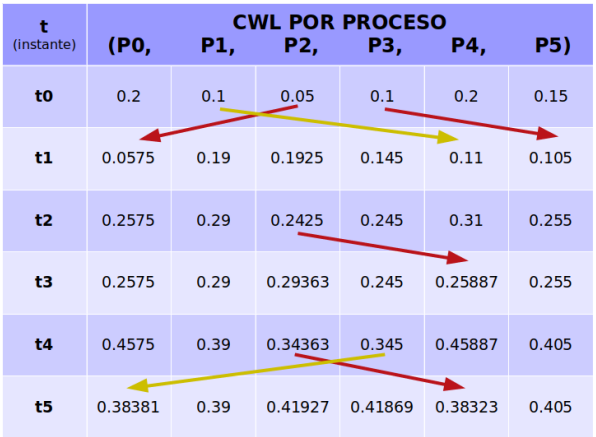


Fig. 2
EJEMPLO DE DISTRIBUCIÓN DE CARGA DE TRABAJO

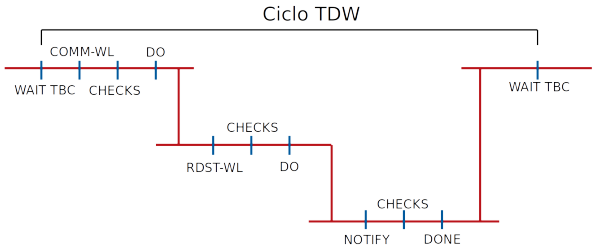


Fig. 3
Ciclo del esquema de comunicaciones

Para ilustrar esto, se establece que los instantes pares (t0, t2, t4) corresponden al avance computacional de cada proceso y los instantes impares (t1, t3, t5) al estado de la carga de trabajo tras las redistribuciones oportunas.

En un primer momento (t_0), la diferencia entre el proceso con mayor CWL (P0) y el proceso con menor CWL (P2) supera el umbral $WR_{diff} = 0.05 \leq 0.15 = CWL_0 - CWL_2$. Por tanto, se produce una redistribución del trabajo, tal que P2 envía parte de su carga a P0. En concreto, envía un 14.25 %, que viene de $min[0.5\ ,\ (CWL_0 - CWL_2) \cdot (1 - CWL_2) = 0.1425]$. De manera análoga, ocurre lo mismo entre P4 y P1, ya que la diferencia de carga de trabajo entre estos supera el umbral, por lo que P1 envía parte de su carga de trabajo a P4. También entre P5 y P3, donde P3 envía parte de su carga de trabajo a P5.

En el instante t_1, se puede ver como queda la carga de trabajo de los procesos tras la redistribución para, en el instante t_2, evolucionar tras otra etapa de computación. A continuación, se produce otra redistribución. En este caso, únicamente la diferencia de carga de trabajo entre P4 y P2 supera el umbral $WR_{diff} = 0.05 \leq 0.0675 = CWL_4 - CWL_2$.

El proceso se repite de la misma manera y, al llegar al instante t_4, la carga de trabajo se distribuye entre P4 y P2 y entre P0 y P3. En caso de seguir desarrollando la evolución, se repetiría la misma mecánica hasta que el trabajo hubiese terminado.

B. Esquema de comunicaciones

En la figura 3 se ilustra el esquema de comunicaciones completo, asumiendo que se produce al menos una redistribución de la carga de trabajo.

En primer lugar, se espera un tiempo *TBC* (*Time Between Communications*). A continuación, los distintos procesos se comunican su carga de trabajo mediante la colectiva no bloqueante *MPI_Iallgather*, al llegar al momento marcado como COMM-WL. Después, se solapan computaciones con comprobaciones del estado de la colectiva no bloqueante (CHECKS) hasta que, finalmente, se comprueba que la comunicación ha terminado. Llegados a este punto, si no fuese a tener lugar ninguna redistribución, el ciclo terminaría y se esperaría un tiempo *TBC* antes de volver a comenzar y comunicar la carga de trabajo. No obstante, en caso de que la configuración de las cargas de trabajo dé lugar al menos a una redistribución, se procedería a realizar la segunda etapa.

En la segunda etapa, los procesos que participan en alguna redistribución llegan al momento marcado como RDST-WL. Aquí, los procesos que van a enviar carga de trabajo comienzan una comunicación punto a punto no bloqueante como emisores, utilizando la función *MPI_Isend*. Por otro lado, los procesos que van a recibir carga de trabajo hacen lo propio como receptores, utilizando la función *MPI_Irecv*. En cuanto a aquellos procesos que no van a participar en ninguna redistribución, saltan directamente a la tercera etapa. Volviendo a los procesos implicados en la redistribución, solapan computación con comprobaciones sobre el estado de la transmisión de carga de trabajo (CHECKS), hasta que comprueban que la comunicación ha terminado y avanzan a la tercera etapa.

Finalmente, al llegar a la tercera etapa, todos los receptores envían una notificación a todos los demás procesos para indicar que han terminado de recibir la carga de trabajo (NOTIFY), utilizando para ello la colectiva no bloqueante *MPI_Ibcast*. Todos los procesos realizan comprobaciones por cada notificación que esperan recibir (CHECKS), hasta que tengan constancia de que han terminado todas las redistribuciones pendientes. Al término de esta etapa, se considera el ciclo concluido y se espera un tiempo *TBC* antes de comenzar el siguiente, aprovechando el periodo de espera para computar sin atender a las comunicaciones.

Cabe mencionar que todos los procesos llevan cuenta en un vector *RIP* (*Redistributions In Process*) de las notificaciones que deben recibir. Esto es necesario porque el vector *RIP* indica en que momento se ha terminado la redistribución, mientras que el *MPI_Request* asociado a la colectiva sirve para comprobar cuando se ha completado la notificación.

En el caso del receptor, tan pronto como termina de recibir la carga de trabajo actualiza el valor en la posición correspondiente del vector *RIP*, sin que los otros procesos dispongan del valor actualizado hasta que termine la comunicación. Sin embargo, el ciclo no debe terminar hasta que todos los procesos tengan el vector *RIP* actualizado, lo cual sólo se puede establecer con seguridad cuando se cumplen las dos condiciones que siguen:

1. Todos los valores de *RIP* están a 0.
2. Todas las *MPI_Request* asociadas a notificaciones han terminado, es decir, han sido consumidas por la función *MPI_Test*.

En la figura 4, el instante t_0 corresponde al momento en que P0 termina de recibir parte de la carga de trabajo de P1 y, por consiguiente, actualiza su vector *RIP* oportunamente. No obstante, P1, P2 y P3 todavía no han recibido la notificación y no tienen su vector *RIP* actualizado. Si no se requiriese la doble comprobación para terminar el ciclo, P0 consideraría que ha concluido y comenzaría la espera del siguiente ciclo antes de que los otros procesos hu-

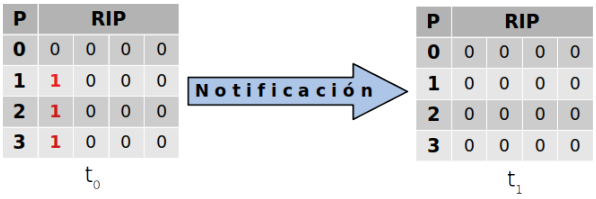


Fig. 4
Tabla RIP

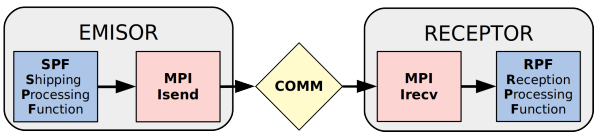


Fig. 5
Gestión del envío y recepción de la carga de trabajo

biesen terminado el ciclo anterior. Por tanto, todas las comunicaciones no bloqueantes se sincronizan en cierta manera al retrasar el comienzo del siguiente ciclo hasta que todos los procesos hayan terminado el anterior. Como puede verse en el instante t_1, que tiene lugar una vez ha concluido la notificación, todos los procesos han constatado que se ha terminado el ciclo y pueden comenzar con la espera que marca el comienzo del siguiente.

C. Caso de uso

El algoritmo de carga de trabajo dinámica temporizada se ha implementado de tal manera que, para aplicarse a un código MPI que realiza computaciones en un bucle, baste con cumplir el pseudocódigo del algoritmo 2.

Algorithm 2 Algoritmo de carga de trabajo dinámica temporizada

```
conf ← adequate settings
TDW_start(conf)
for i = 0 to n do
  if conf.modulo ≡ 0 mod n then
    TDW_balance(CWL)
  end if
  compute(i)
  while i=n-1 and TDW_todo() do
    TDW_balance(CWL)
  end while
end for
```

El primer paso consiste en establecer los parámetros de configuración. En el segundo paso, la configuración es utilizada para iniciar el algoritmo, justo antes de comenzar el bucle distribuido (*TDW_start*). Al comienzo del bucle, se comprueba si la iteración es congruente con cero en el módulo especificado y, en caso de serlo, se invoca a la función *TDW_balance*. Dicha función se encarga de comprobar si ha transcurrido el tiempo necesario y, en caso afirmativo, inicia el proceso de redistribución, tal y como se ha explicado en las subsecciones IV-A y IV-B. En la parte final del bucle se ejecuta otro lazo anidado que captura el comportamiento tras procesar la última iteración, mientras quede trabajo pendiente. Este bucle es necesario para garantizar que todos los procesos terminen y evitar que se den escenarios de espera indefinida. Tal situación podría producirse si algún proceso continuase su flujo de ejecución tras haber terminado su trabajo, mientras los demás esperan indefinidamente por su mensaje para completar sus comunicaciones.

En cuanto a la configuración, además de los parámetros ya explicados, permite especificar dos funciones que se ilustran en la figura 5:

- *SPF* (*Shipping Processing Function*). Esta función se encarga de realizar los procesamientos oportunos para preparar el envío. Su invocación tiene lugar justo antes de la función *MPI_Isend* y se vale de 6 parámetros:
  1. Buffer de envío. Contiene los datos relativos a la carga de trabajo que se delega.
  2. Tamaño. Máxima cantidad de bytes que soporta el buffer de envío.
  3. Destino. Rango del proceso receptor.
  4. *CWL* del emisor. Carga de trabajo completada del proceso que envía la carga de trabajo.
  5. *CWL* del receptor. Carga de trabajo completada del proceso que recibirá la carga de trabajo.
  6. Argumentos extra. Este parámetro se deja a disposición del usuario, para que pase a la función todo aquello que considere oportuno para procesar el envío.
- *RPF* (*Reception Processing Function*). Esta función se encarga de aplicar los procesamientos oportunos a los datos recibidos. Su invocación tiene lugar justo después de la terminación de la recepción asociada a la función *MPI_Irecv* y se vale de 3 parámetros:
  1. Buffer de recepción. Contiene los datos relativos a la carga de trabajo que se asume.
  2. Tamaño. Máxima cantidad de bytes que soporta el buffer de recepción.
  3. Argumentos extra. Este parámetro se deja a disposición del usuario, para que pase a la función todo aquello que considere necesario para procesar la recepción.

V. Análisis de rendimiento

Se ha realizado una comparativa entre las distintas técnicas aplicadas, ejecutando el software con un número de procesos desde 1 hasta 16, obteniendo el tiempo de ejecución en cada caso.

Como puede observarse en la figura 6, la técnica de balanceo de carga dinámico (TDW) presenta menor tiempo de ejecución y, por tanto, un mayor speed-up que las otras dos técnicas analizadas. En cuanto a la distribución estática de la carga de trabajo, resulta evidente que la distribución cíclica (RR) ofrece

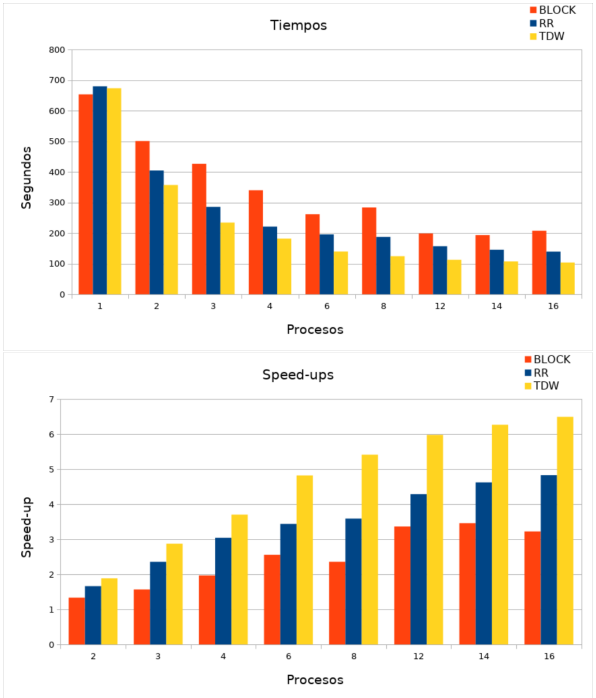


Fig. 6
Rendimiento en 1 nodo

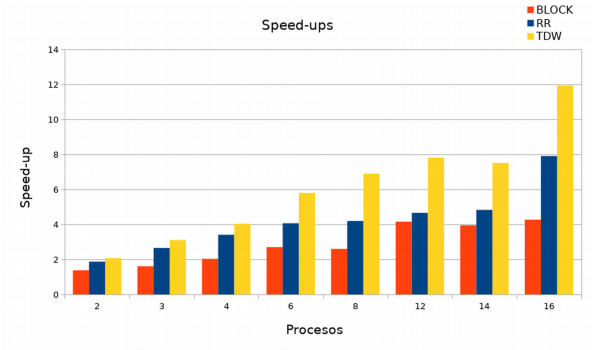


Fig. 7
Rendimiento con 4 procesos por nodo

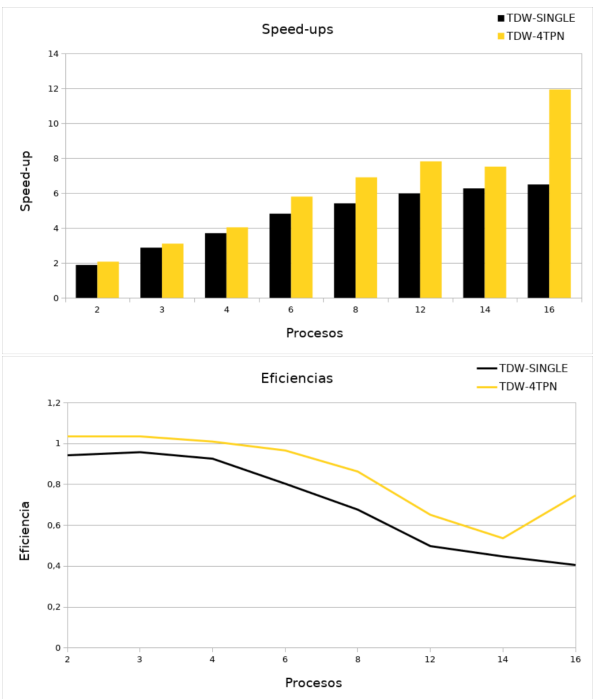


Fig. 8
Rendimiento para 4 procesos por nodo y para un solo nodo

mejores resultados que la distribución por bloques (BLOCK).

Si se atiende al comportamiento de la escalabilidad se puede observar que, aun en el mejor de los casos con 16 procesos, el tiempo más reducido es de 103.71 segundos, que supone un speed-up de 6.49. A raíz de esto, se decide estudiar distintas configuraciones, pues se sospecha que puede estar ocurriendo un problema de congestión del bus de memoria [15]. Para ello, se vuelven a realizar las ejecuciones configurando un límite de 4 procesos por nodo. De esta manera, al utilizar 16 procesos no se ejecutarán todos en el mismo nodo, sino que habrá 4 procesos en 4 nodos. Los resultados correspondientes a esta configuración pueden verse en la figura 7 y suponen un aumento general del rendimiento para todas las estrategias. En términos de speed-up, esto implica que el mejor valor que se obtenía (6.49) al ejecutar 16 procesos en un solo nodo suba hasta alcanzar una cota de 11.94 al ejecutar 16 procesos con un máximo de 4 procesos por nodo.

Como se puede ver en las gráficas de la figura 8, la eficiencia de la técnica varía según la configuración de la infraestructura computacional. La hipótesis de que estaba ocurriendo un problema de congestión del bus de memoria parece confirmarse, ya que encaja con la evolución a un mejor rendimiento que se obtiene al distribuir los procesos entre varios nodos. Otra posible solución a este problema son los algoritmos conscientes de la congestión [16], que tratan de encontrar un equilibrio entre la optimización de la localidad de los datos y la congestión del bus de memoria.

Retomando el análisis de rendimiento se observa que, limitando a un máximo de 4 procesos por bus de memoria, se consiguen mejores resultados. Esto es bastante significativo, ya que se asume un sobrecoste derivado de las comunicaciones, que ya no ocurren dentro del mismo nodo, sino que deben realizarse a través de la red InfiniBand[14] del FinisTerrae-II.

Por último, tanto para terminar de contrastar la hipótesis, como para poner a prueba la escalabilidad de la mejor solución, se han llevado a cabo ejecuciones hasta un máximo de 64 procesos. Como puede verse en las gráficas de la figura 9, la configuración de 2 procesos por nodo alcanza su mejor rendimiento con 36 procesos. A partir de este número de procesos no se consigue un mejor rendimiento, ni siquiera a costa de la eficiencia.

En cuanto al caso de 4 procesos por nodo, su rendimiento a partir de los 16 procesos es peor que el

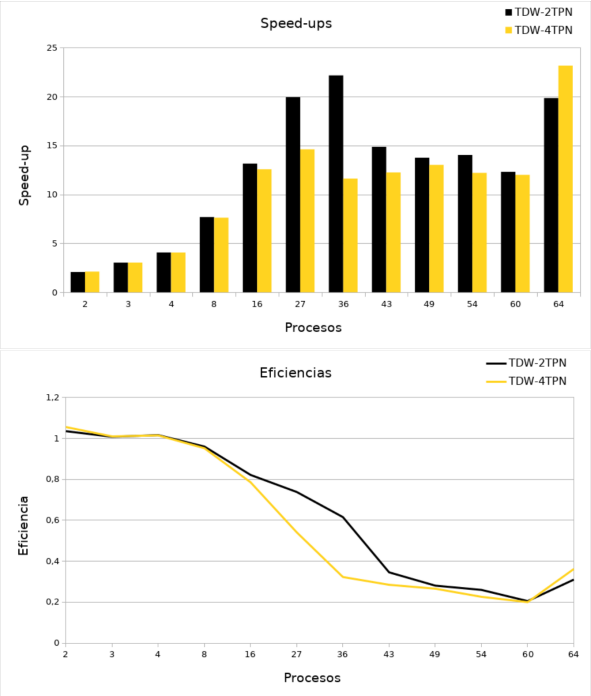


Fig. 9
Rendimiento para 4 y 2 procesos por nodo

obtenido con la configuración de 2 procesos por nodo. No obstante, al llegar a los 64 procesos tarda 28.88 segundos, frente al mejor tiempo de la configuración de 2 procesos por nodo (30.32 segundos). Sin embargo, la eficiencia es peor en comparación, a causa de que esa ligera mejora en el tiempo requiere de un número más elevado de procesos. Así pues, se gastan muchos más recursos computacionales (64 entidades computacionales, en este caso 64 cores en 16 procesadores) para obtener un resultado prácticamente equivalente al que se consigue con una configuración de 2 procesos por nodo (36 entidades computacionales, distribuidas en 36 cores de 18 procesadores).

VI. Conclusiones

Se ha desarrollado una aplicación paralela para la caracterización de elementos viales con LiDAR terrestre en la que la rutina que necesita mayor tiempo de ejecución puede mejorar su rendimiento significativamente a través de un balanceo de la carga computacionalmente eficiente. En este trabajo se presenta una estrategia que mejora el rendimiento para este tipo de problemas.

En cuanto a los métodos de paralelización propuestos, la técnica de balanceo dinámico de la carga de trabajo ha conseguido el menor tiempo de ejecución, el mayor speed-up y la mejor eficiencia. Por tanto, puede concluirse que ha sido el mejor método de paralelización de entre todos los estudiados.

Se ha probado que adaptar el programa a las características computacionales de su contexto de ejecución puede afectar más que significativamente al rendimiento. Concretamente, se obtiene un mejor rendimiento al limitar el número de procesos por nodo, lo que se explica por la reducción de la congestión del bus de memoria en los procesadores.

Por último, se considera de interés, como vía de trabajo futuro, utilizar las señales del sistema operativo [17] para mejorar el algoritmo de balanceo dinámico. Proponiendo sustituir la espera activa por una señal que notifique al proceso cuando tenga que realizar la comprobación oportuna.

Agradecimientos

Este trabajo fue financiado en parte por Babcock International Group PLC (Civil UAVs Initiative de la Xunta de Galicia), la Dirección General de Tráfico (Proyecto BIG-GEOMOVE, SPIP2017-02340), el Ministerio de Educación, Cultura y Deporte, [Proyecto TIN2016-76373-P], la Xunta de Galicia [Proyectos GRC R2016/045 y R2016/037], la Consellería de Cultura, Educación e Ordenación Universitaria (acreditación 2016-2019, ED431G/08, ED431C 2018/2019) y la Unión Europea (European Regional Development Fund - ERDF).

Así mismo, el análisis de rendimiento de las distintas técnicas y configuraciones fueron posibles gracias al Centro de Supercomputación de Galicia (CESGA), que ofreció su infraestructura computacional para la realización de este trabajo.

Referencias

[1] Centro de Supercomputación de Galicia (CESGA), `http://cesga.es`
[2] Thomas Rauber, Gudula Rünger, *Parallel Programming for Multicore and Cluster Systems* Springer, Berlin, Heidelberg, 2013
[3] Peter S. Pacheco, *Parallel Programming with MPI* Morgan Kaufmann Publishers, Inc., 1997
[4] Message Passing Interface Forum *MPI: A Message-Passing Interface Standard* , June 4, 2015
[5] William Gropp, Torsten Hoefler, Rajeev Thakur, Ewing Lusk, *Using Advanced MPI: Modern Features of the Message-Passing Interface* The MIT Press; 1 edition (November 7, 2014)
[6] Antonio Lain, Prithviraj Banerjee, *Techniques to overlap computation and communication in irregular iterative applications* ICS '94 Proceedings of the 8th international conference on Supercomputing, pp. 236-245, 1994
[7] Torsten Hoefler, Andrew Lumsdaine, Wolfgang Rehm, *Implementation and performance analysis of non-blocking collective operations for MPI* SC '07 Proceedings of the 2007 ACM/IEEE conference on Supercomputing, 52, 2007
[8] Taher Saif, Manish Parashar *Understanding the Behavior and Performance of Non-blocking Communications in MPI* Euro-Par 2004 Parallel Processing, pp 173-182, 2004
[9] NOAA. *What is LIDAR?* website, `https://oceanservice.noaa.gov/facts/lidar.html`, 06/25/18.
[10] J. Shan, C.K. Toth, *Topographic laser ranging and scanning: principles and processing* CRC press, 2008
[11] Haala, Norbert and Peter, Michael and Kremer, Jens and Hunter, Graham, *Mobile LIDAR Mapping for 3D Point Cloud Collection in Urban Areas - A Performance Test* In: Proceedings of XXI ISPRS Congress, Beijing, China, July 3-11, 2008
[12] Brent S. SCHWARZ, James A. HASLIM, Nicholas M. ITURRARAN, Michael D. KARASOFF *LiDAR scanner* Uber Technologies Inc, US9470520B2, 2013
[13] C. A. R. Hoare, *Algorithm 63, Partition; Algorithm 64, Quicksort* Communications of the ACM, Vol. 4, p. 321, 1961.
[14] Pfister, G. F, *An Introduction to the InfiniBand™ Architecture. High Performance Mass Storage and Parallel I/O* (pp. 617-632) Austin, Texas: IBM Enterprise Server Group.

[15] John L. Hennessy, David A. Patterson, *Computer Architecture: A Quantitative Approach* Morgan Kaufmann Publishers, Inc., pp. 238-253, 2007

[16] Mulya Agung, Muhammad Alfian Amrizal, Kazuhiko Komatsu, Ryusuke Egawa and Hiroyuki Takizawa, *A Memory Congestion-aware MPI Process Placement for Modern NUMA Systems* IEEE 24th International Conference on High Performance Computing (HiPC), 2017

[17] Michael Kerrisk, *signal(7)* Linux Programmer's Manual 3.22, The Linux Kernel Archives, 2009

Reorganización de matrices en algoritmos de barrido radial sobre Modelos Digitales del Terreno

Andrés Jesús Sánchez[1], Luis F. Romero[1], Siham Tabik[2], Gerardo Bandera[1]

Resumen— **Es muy frecuente, en los sistemas de información geográfica que trabajan con modelos digitales del terreno, el uso de algoritmos de barrido radial para el estudio de variables asociadas a parámetros cuya magnitud decrece con el cuadrado de la distancia, como las señales de radio, las ondas de sonido, o la propia visión humana. Sin embargo, dichos algoritmos están asociados a un acceso a las matrices de datos que, en la mayoría de los casos, aun siendo regular, derivan en un mal aprovechamiento de la localidad de la memoria. En este trabajo se muestra cómo la completa reorganización previa de las matrices de datos, en función de la dirección radial que corresponde, produce una considerable mejora del rendimiento, especialmente en algoritmos de elevada intensidad computacional. Sirva como ejemplo el cálculo de cuencas visuales totales, que es utilizada en este trabajo como caso de estudio. Por otra parte, la reestructuración matricial propuesta abre la puerta al uso intensivo de GPUs en muchos algoritmos para los que nunca se han considerado, por su irregularidad y baja eficiencia.**

Palabras clave— **Modelos Digitales del Terreno, arquitecturas heterogéneas CPU-GPU, Supercomputación, Visibilidad Total.**

I. Introducción

SON muy numerosos los algoritmos en los que el análisis de un Modelo Digital del Terreno, o MDT, se realiza mediante un barrido radial desde un punto situado en el propio territorio. Normalmente, a dicho punto de referencia se le denomina Punto de Observación, o POV (acrónimo del inglés *point of view*), ya que se considera que dicho punto tiene un especial interés. En muchos casos, el POV es emisor o receptor de señales cuya intensidad decrece con el cuadrado de la distancia (una señal emisora de radio, la visión de un observador sobre el terreno, o una fuente de sonido). Precisamente debido al decaimiento de la intensidad de las señales, este tipo de problemas se caracterizan porque la relación entre el POV y el resto del territorio se debilita con la distancia y, en consecuencia, la precisión requerida es más baja para los puntos más alejados.

Los algoritmos de barrido radial analizan la relación entre el POV y el resto del territorio mediante una discretización acimutal de los cálculos. Es decir, considerando al POV como vértice, el territorio se divide en sectores angulares horizontales, de forma similar a los cuadrantes de una rosa de los vientos. E independientemente del número de sectores (ya sean los 360 grados sexagesimales, o los 4 cuadrantes N, S, E y W) en todos estos problemas se considera que el eje del sector es el representante estadístico de todo el sector. Es obvio que la representatividad de dicho eje se reduce a a medida que nos alejamos del vértice, ya que el ancho del sector aumenta linealmente con el radio. Pero también resulta evidente que se reduce en la misma medida la necesidad de precisión, haciendo que este tipo de algoritmos sean especialmente adecuados para estos problemas. De hecho, algoritmos similares se pueden extender a problemas tridimensionales, como pueden ser las técnicas de *ray-tracing* en visión por computador, o en el renderizado de volúmenes.

A. Visibilidad del territorio

Quizás uno de los problemas más representativos de lo expuesto anteriormente sea el cálculo de la visibilidad de un observador ubicado a cierta elevación sobre el terreno. Un conocimiento preciso de la visibilidad en un territorio proporcionaría una herramienta de gran interés para distintas aplicaciones en diversos campos, tales como las telecomunicaciones, planificación del medio natural, ecología, turismo, arqueología, etc. Así, como ejemplo, la elaboración de algoritmos que determinen el menor número posible de puntos de observación necesarios para dar una máxima cobertura visual (o de telefonía) de un terreno, se simplificaría enormemente si se conociera a priori qué parte del territorio es visible desde cada punto [1].

Se conoce como Cuenca Visual (o *viewshed, VS*, del inglés) de un observador del terreno, al área del territorio visible desde un determinado punto que, normalmente, se encuentra situado a una altura h sobre el terreno. Existen numerosos algoritmos que calculan el VS desde un único punto de vista o, como mucho, desde un número muy reducido de puntos de observación. En este contexto, la mayoría de sistemas de información geográfica (GIS, por sus siglas en inglés) como ArcGIS [2], o GRASS GIS [3], e incluso aplicaciones de consumo masivo, como Google Earth, incorporan módulos para el cálculo de la cuenca visual. Además de ellos, existen numerosos algoritmos con el mismo propósito [4], [5], [6], [7], [8].

El algoritmo responsable del cálculo de la visibilidad mediante un barrido radial es bastante simple. Como se ha expuesto anteriormente, es necesaria una discretización previa del territorio que rodea al observador en un número ns de sectores acimutales (en lo sucesivo, utilizaremos sectores de 1^o y $ns = 360$

[1]Dpto. de Arquitectura de Computadores, Universidad de Málaga, e-mail: ajsanchez@ac.uma.es, felipe@uma.es, gbandera@uma.es

[2]Dpto. Ciencias de la Computación e Inteligencia Artificial, Universidad de Granada, e-mail: siham@ugr.es

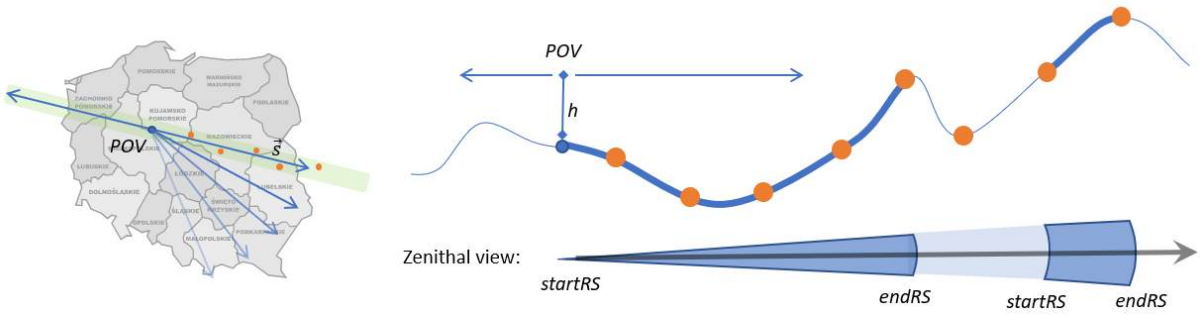


Fig. 1. Barrido radial para calcular la visibilidad.

para simplificar). Para cada uno de los sectores s elegidos, y para cada sentido del eje que pasa por el sector, se calcula su visibilidad seleccionando puntos próximos al eje como representantes de cada sector ($selectAxisPointSet$, suponemos el MDT representado con una malla regular cartesiana):

Algoritmo 1 Cálculo de la cuenca visual de un punto POV.

```
function viewshed(i, j, POVheight)
global point POV = MDT[i, j];
POV.h += POVheight;
pointSet axis = selectAxisPointSet(POV, s);
float VS = 0;
for s = 0, (ns/2 − 1) do
  VS += linearViewshed(axis, true); //forward
  VS += linearViewshed(axis, false);//backward
end for
VS *= (π/ns); //Papus theor. scaling
endfunction
```

donde i, j y $POVheight$ representan las coordenadas y elevación del punto de observación POV (el observador se sitúa ligeramente por encima del terreno); $axis$ es el conjunto de puntos que representan al sector (normalmente próximos al eje, según se muestra en rojo en la figura 1) y VS es la cuenca visual.

La cuenca visual (normalmente se mide la superficie visible) en un determinado sector, y para un observador situado sobre el eje del sector, se obtiene calculando la visibilidad de cada uno de los puntos restantes del eje, mediante un recorrido desde el más próximo al más lejano en el sentido correspondiente:

Algoritmo 2 Cálculo de VS para un punto en el eje de un sector.

```
function linearViewshed(axis, forward)
global bool visible = true;
global float max = -∞; // Max. angle
global pointSet visibleSet = emptySet;
T = axis.POV
while T != (forward)?axis.last : axis.first do
  pointViewshed(axis.POV,T);
  T = (forward)?axis.next() : axis.prev();
end while
return visibleSet.measure();
endfunction
```

donde $visibleSet$ es el conjunto de puntos del eje que son visibles desde POV, y T el punto destino. Obviamente, un punto destino es visible desde el punto origen si su altitud angular es superior a la de todos los puntos más próximos, y que han sido previamente recorridos y analizados. Sin embargo, es frecuente utilizar la tangente, en lugar del propio ángulo, para evitar la ejecución de algunas operaciones trigonométricas costosas e innecesarias, como se presenta en el Algoritmo 3.

Algoritmo 3 Cálculo de la visibilidad de un punto T desde POV.

```
function pointViewshed(POV, T)
float dist = distance(POV,T);
float angle = (T.h − POV.h)/dist;
visible = angle > max
if(visible) visibleSet.add(T);
endfunction
```

Una optimización evidente para este algoritmo, que no solo reduce accesos a memoria sino también cálculos, ha sido utilizada en [9], y consiste en almacenar sólo los puntos en los que empieza y finaliza una secuencia en la que todos los puntos son visibles desde POV, y a los que denominan *segmentos visibles* o *anillos sectoriales*, por la figura geométrica que se obtiene al extrapolar el segmento a todo el sector (véase la figura 1). En este caso, el algoritmo quedaría de esta forma:

Algoritmo 4 Visibilidad de T desde P mediante anillos sectoriales.

```
function pointViewshed(POV, T)
float dist = distance(POV,T);
float angle = (T.h − POV.h)/dist;
bool prevVisible = visible
visible = angle > max
bool startRS = !prevVisible & visible;
if(startRS) dist0 = dist;
bool endRS = prevVisible & !visible;
if(endRS) VS += dist * dist − dist0 * dist0;
endfunction
```

donde $startRS$ y $endRS$ son las variables que se encargan de indicar si durante el barrido a lo largo del eje se ha encontrado o abandonado una secuencia de puntos visibles. Por otro lado, se ha añadido una

variable global *dist*0 en la que se almacena la distancia entre POV y el primer punto de un tramo visible. Este valor se utiliza para calcular la superficie del anillo sectorial cuando aparezca el último punto del tramo, mediante la diferencia de cuadrados. Recordemos que el área de un anillo sectorial de 1º de apertura es $(\pi/360) \cdot (R^2 - r^2)$.

II. Complejidad de los algoritmos de barrido en los MDT

Los algoritmos de barrido sectorial pueden reducir significativamente la complejidad de los cálculos inherentes a un determinado problema. Si, por ejemplo, intentamos calcular la elevación media de un territorio, podríamos reducir la complejidad desde $O(\mathrm{N})$ a $O(\mathrm{s}{\cdot}\mathrm{N}^{1/2})$, siendo N el tamaño del MDT, medido en puntos. Teniendo en cuenta que los MDT superan ampliamente los varios millones de puntos y que la discretización sectorial en raras ocasiones supera el grado de precisión, estamos hablando de una reducción en las operaciones que oscila entre uno y tres órdenes de magnitud [10]. Para el cálculo de la visibilidad, y teniendo en cuenta que, a priori, cualquier punto del territorio podría ser obstáculo para la visibilidad entre dos puntos cualesquiera del mismo, la complejidad del problema (ya optimizado) es $O(\mathrm{N}^{1.5})$, mientras que la del algoritmo basado en la discretización radial, como hemos visto, es de $O(\mathrm{s}{\cdot}\mathrm{N}^{0.5})$.

Aún así, estamos hablando de un número bastante elevado de operaciones que hacen que la supercomputación o el paralelismo sean altamente recomendables para estas aplicaciones. De hecho, uno de los problemas que hasta ahora se consideraban inabordables es el de la visibilidad total, que describiremos a continuación.

A. Visibilidad total de un MDT

Hasta hace apenas 5 años, nunca se había considerado abordable el problema del cálculo de la cuenca visual desde todos los puntos del territorio representado por un MDT (es decir, con los N puntos de observación). El motivo era evidente: en el mejor de los casos, el cálculo del VS de un solo POV en un territorio, representado por unos pocos millones de puntos, ya requería, al menos, de varios segundos e incluso minutos de CPU.

Si el MDT está representado por una malla cartesiana de puntos de dimensiones *dimx* y *dimy*, el problema se podría expresar de la siguiente forma:

Algoritmo 5 Cálculo de la cuenca visual total.

```
for i = 0, dimy − 1 do
  for j = 0, dimx − 1 do
    tvs[i,j] = viewshed(i, j, POVheight)
  end for
end for
```

donde *tvs*, del inglés *total viewshed* es una matriz, con las mismas dimensiones del MDT, en la que se guarda la extensión del VS de cada punto, considerado como POV del territorio. Un ejemplo de los resultados obtenidos se muestran en la figura 2, donde cada celda del MDT rasterizado indica, mediante un mapa de colores, si los puntos del territorio tienen muy baja visibilidad (gama de azules), o ésta es muy elevada (gama de rojos). La complejidad inherente del problema es en este caso, de $O(\mathrm{N}^3)$, ya que resolvemos N veces un problema de $O(\mathrm{N}^2)$, mientras que la del algoritmo basado en la discretización radial subiría hasta $O(\mathrm{s}{\cdot}\mathrm{N}^{1.5})$, lo que justifica que nunca se hubiera abordado (o al menos, con MDTs de grandes dimensiones).

Sin embargo, trabajos recientes han demostrado que, mediante técnicas *multigrid*, y ciertas optimizaciones de memoria es posible hacer los cálculos de la visibilidad total para MDTs de gran tamaño en tiempos razonables [11], [12], demostrándose incluso la enorme utilidad de dichos algoritmos para la ubicación de torres de observación [13]. Aún así, estos algoritmos trabajan con cifras descomunales para los accesos a datos y las operaciones en punto flotante, por lo que solo se ha podido tratar adecuadamente el problema gracias a una gestión eficiente de los accesos a memoria.

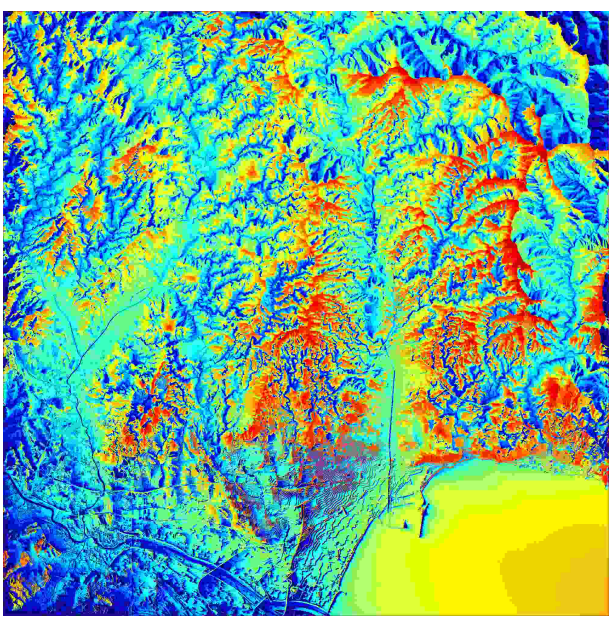


Fig. 2. Mapa de cuencas visuales totales en el entorno del municipio de Málaga, España.

B. Reutilización de datos en la banda de visibilidad

El conjunto de puntos que se tienen en cuenta para calcular la cuenca visual (representados por la estructura *axis* en el algoritmo 1) es denominado por los autores Cervilla *et al*[14] como Banda de Visibilidad, o BoS (del inglés *band of sight*). En otros algoritmos (véase, por ejemplo, Franklin *et al*[1]) se utilizan conceptos similares, aunque siempre hacen referencia a los puntos próximos al eje, o al resultado de la interpolación de los mismos. En la figura 1, a la izquierda, la banda de visibilidad se muestra en color verde. Es obvio que la reutilización de los datos en la banda de visibilidad, en la que concurren los puntos del MDT

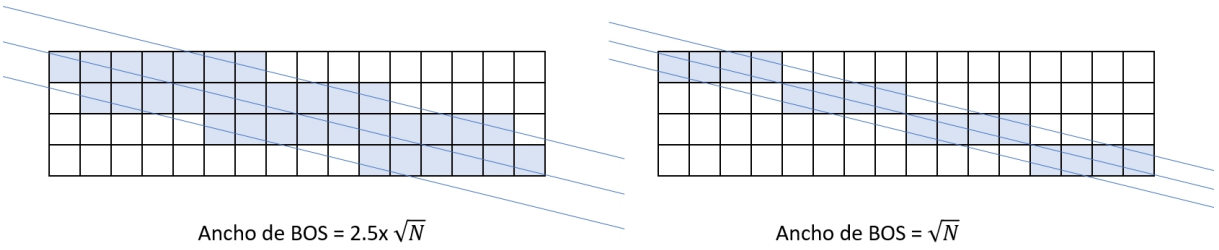


Fig. 3. Nodos que contribuyen a la BOS, con distintos anchos para la banda.

alineados en la dirección del sector elegido, es la clave de la optimización de los algoritmos de visibilidad total, pues precisamente se puede reutilizar dicha estructura para calcular la cuenca visual de todos los puntos alineados en la dirección de un mismo sector. En ese sentido se produce la principal aportación de dichos autores en [11]. Como se ha mencionado anteriormente, consiste en el máximo aprovechamiento de la memoria. Para ello, su algoritmo garantiza que sólo se va a producir un fallo de cache antes de ejecutar todas las operaciones aritméticas implicadas en el cálculo de la visibilidad de un único punto en un determinado sector. Para alcanzar esa considerable reutilización de los datos, han invertido las operaciones: en lugar de seleccionar un punto y calcular su cuenca visual, que sería la forma más natural de proceder (según se propone en el algoritmo 5), han intercambiado los bucles para que en primer lugar se seleccione el sector, y en segundo lugar se realice el barrido de puntos (Stewart, [10]). Gracias a ello, han conseguido resultados muy sorprendentes, que no son más que una consecuencia de la elevada intensidad operacional de su algoritmo que, para un modelo digital extenso (que cubre la superficie de Andalucía, con 10x10m de precisión), y en el que el número promedio de anillos sectoriales en cada dirección es 12, resulta ser IO=257 $flops/byte$ [14].

Sin embargo, el algoritmo que proponen tiene una clara limitación: la altísima secuencialidad del bucle que hace el barrido sobre los puntos. Sin entrar en los complicados detalles de la gestión de la estructura BoS, se puede afirmar que es imposible averiguar el estado de la estructura —para un punto de observación POV— sin conocer el valor de la estructura en el punto anterior. Esto no supone un impedimento para explotar sobradamente el paralelismo en multicomputadores o multiprocesadores, ya que la propia parcelación del territorio, por un lado, y el procesamiento sectorial, por otro, proporcionan miles de hebras que pueden ejecutarse en paralelo. Sin embargo, los experimentos llevados a cabo en GPUs y Xeon-Phi, arquitecturas en las que un grano más fino es posible, no han sido tan satisfactorios [14].

También existen limitaciones a la vectorización. Si se observan los algoritmos 4 y 5, que representan el núcleo principal en el cálculo de la visibilidad, existe una operación de división que es claramente determinante para el coste de CPU. Dado que no se necesita una alta precisión para los cálculos (errores en el resultado de un 20% son perfectamente admitidos [15]), la operación de división podrían ser perfectamente sustituida por el cálculo de recíprocos con baja precisión (por ejemplo, con el juego de instrucciones AVX de intel) vectorizando el lazo del algoritmo 2.

III. Reorganización de la malla de puntos

Obsérvese la figura 3, en la que se muestra una fracción de la banda de visibilidad (BoS, según la nomenclatura de [9]) para un sector $s = 14^o - 346^o$. Para esa dirección, sólo los puntos de la malla en color oscuro son considerados en los cálculos. En algunos algoritmos de barrido sectorial, como en [5], es la distancia al eje la que determina el número de puntos de la banda, mientras que para Cervilla *et al*, es el número de puntos contenidos en la estructura. En la figura se muestran los valores para dos casos particulares para el tamaño de BoS.

El extenso estudio estadístico realizado en [14] demuestra que el tamaño de la estructura no es un factor determinante, mientras sea del orden $N^{0.5}$. Son además muchos los estudios que determinan que en la mayoría de los problemas de barrido radial (y como consecuencia de la pérdida de precisión con la distancia) el método utilizado para la interpolación de los datos del MDT al eje del sector tampoco resulta ser un factor determinante en la precisión de los resultados[1].

A. Resumen de la propuesta

Teniendo en cuenta las limitaciones expuestas, en este trabajo se propone una completa reorganización de los datos que permita la explotación de todo el paralelismo existente sin perjudicar la precisión de los resultados. En particular, el algoritmo propone:

- Utilizar el método de barrido de Stewart [10], o lo que es lo mismo, colocar el lazo que recorre los sectores en el exterior al ser éste el único modelo que garantiza la reutilización de los datos alineados en una dirección.

[1]Entre los métodos de interpolación que se han considerado en [14] están la interpolación bicúbica, el kriging (llamado así por el nombre de su creador, Daniel Krige), y el algoritmo de proximidad de Bresenham. El kriging se diferencia de otros métodos más habituales (como las interpolaciones bilineal o bicúbica) al asumir que la altitud es una variable regionalizada. Esta hipótesis supone que existe una correlación espacial entre la elevación de un lugar y la de su entorno (esencialmente, que existe un cierto patrón), y por tanto puede deducirse un valor de elevación a partir de los valores del entorno de acuerdo a unas funciones homogéneas en toda el área.

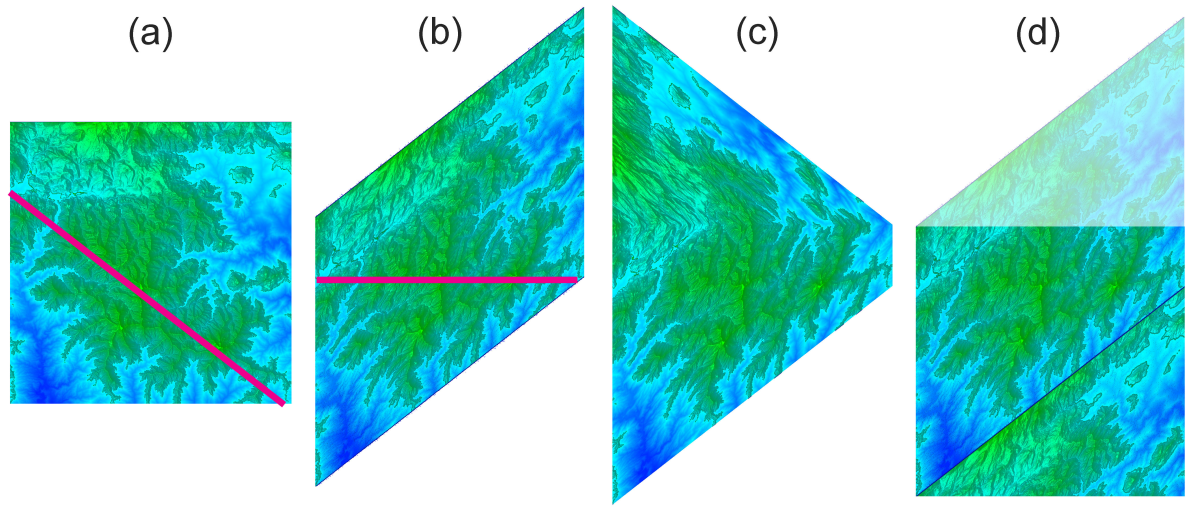


Fig. 4. Fases de redistribución de las matrices: a) Matriz original; b) Matriz reorganizada para 38°; c) Matriz compactada; y d) Matriz compactada para GPU.

- Dada una dirección del sector, construir simultáneamente todas las bandas de visibilidad que atraviesen en paralelo el MDT. Esto implica la creación de un nuevo MDT del territorio y que, por las diferentes longitudes de las bandas, tendría un aspecto más o menos sesgado.
- Simplificar el método de interpolación para el cálculo de las bandas paralelas utilizando el algoritmo de Bresenham, que se usa habitualmente para el rasterizado de líneas. Éste se elige por su velocidad, así como por la suficiente fidelidad para el problema considerado (que no requiere gran precisión en la distancia).
- Compactar la información, para que, en caso de recurrir al uso de GPUs para el procesamiento de los datos, el acceso a la estructura, así como las computaciones realizadas sean lo más regulares posibles, y reduzcan al máximo los condicionales.

El algoritmo resultante tras la reestructuración de la matriz es el siguiente:

Algoritmo 6 Cálculo de la cuenca visual total mediante reestructuración matricial de un MDT.

```
function totalviewshed(POVheight)
float VS = 0;
for s = 0, (ns/2 − 1) parallel do
  float sectorVS = 0;
  sMDT=skew(MDT, s); //rewrite MDT
  compact(sMDT);
  for i = 0, √N − 1 parallel do
    sectorVS+ =linearViewshed(i, true);
    sectorVS+ =linearViewshed(i, false);
  end for
  // Skewing and Papus theo. scaling:
  VS+ = (k ∗ (π/ns) ∗ sectorVS)
end for
endfunction
```

donde la funcion *skew* es la que redistribuye los datos del MDT en una nueva matriz, y k es una constante para cada sector, debido a la deformación del modelo digital sesgado. En el algoritmo, se ha simplificado a $\sqrt{N}$ el tamaño del lazo que recorre las filas de la estructura sesgada, aunque en la práctica este número oscila entre $dimx$ y $dimx + dimy$ (entre 0 y 45°), entre $dimx + dimy$ y $dimy$ (para el rango 45° a 90°), entre $dimy$ y $dimy + dimx$ (para el rango 90° a 135°), y entre $dimy + dimx$ y $dimx$ (para el rango 135° a 180°).

Para visualizar gráficamente el método, la figura 4 muestra la redistribución de filas y columnas de un MDT con las elevaciones correspondientes al Parque Nacional Sierra de la Nieves, en Málaga. A la izquierda (a) se muestra la matriz original de datos (supondremos en éste ejemplo que, en memoria, se almacenan contiguos los datos de una misma latitud; es decir, el lazo externo va de norte a sur, y el interno de oeste a este).

Supongamos que se ha elegido la dirección 128°-322° para el cálculo de la cuenca visual total. La figura 4(b) muestra gráficamente el modelo sesgado (representado por la estructura sMDT en el algoritmo 6). Utilizando el algoritmo de Bresenham, se han proyectados todos los segmentos paralelos de la figura 4(a) en 4(b) de forma que el tamaño de ambas estructuras (el número de elementos no nulos) es idéntico. En la nueva matriz, a diferencia de la original, todos los datos en la dirección de búsqueda están en la misma fila y, por lo tanto, el acceso a memoria es secuencial.

Tras la reescritura de la matriz, ésta puede ser compactada de dos formas: en la figura 4(c), todos los datos de la matriz sesgada se han desplazado a la primera columna. Por otro lado, en la figura 4(d), además, se han reubicado los datos en otra columna, con el objetivo de compactar aún más la información (el triángulo del territorio en color claro se ha eliminado y aparece reubicado abajo). Obsérvese que en la figura 4(d), muchos algoritmos, como TVS pueden operar sin la necesidad de condicionar la longitud de la fila.

IV. Resultados

Se han comparado los resultados obtenidos en el cálculo de la cuenca visual total entre el algoritmo de Cervilla *et al* (*tvs*) y el que se propone en este trabajo, *sdem*, del inglés *skewed digital elevation model*. Se consideran, para los cálculos, observadores situados a 1.5 metros de elevación sobre un MDT de la Sierra de las Nieves de 2000x2000 puntos y 10x10m de precisión. Se ha elegido esta zona por su representatividad, ya que contiene zonas muy llanas, y otras muy montañosas y escarpadas. Los cálculos se han ejecutado en tres sistemas multiprocesador basados en los procesadores Intel Xeon E5-2698v3, a 2.30GHz, Intel core I5-6500, a 3.20GHz, y AMD Ryzen5-2600 a 3.4GHz, con 32, 8 y 6 cores respectivamente.

En el caso del algoritmo *tvs* se ha elegido un tamaño $\sqrt{N}+1$ para la banda, de forma que la estructura BoS coincide prácticamente con la interpolación de Bresenham (véase la figura 3, derecha). Por este motivo, los resultados obtenidos con ambos algoritmos son numéricamente idénticos.

En lo que respecta al número de cores, se ha aplicado paralelismo al bucle externo (que barre los sectores) y en ambos casos la escalabilidad es lineal (como cabía esperar, ya que los sectores son completamente independientes y los cálculos asociados a los distintos sectores tienen carga muy equilibrada, por motivos estadísticos).

Respecto al tiempo empleado por cada core en cada sector, que al fin y al cabo es el parámetro que realmente mide la calidad y eficiencia de los algoritmos propuestos, las tres arquitecturas ofrecen tiempos de ejecución que rondan los 60 segundos para el algoritmo *tvs*, de los que aproximadamente el 10% se emplea en el reordenamiento de puntos. Por su parte, el algoritmo *sdem* requiere menos de un segundo para redistribuir la matriz, y reduce en más de un 25% el tiempo de cálculo del sector (bajando hasta 30-35 segundos, según la arquitectura), debido sobre todo a la regularidad del lazo que recorre la banda de visibilidad.

V. Conclusiones

Los resultados obtenidos en este trabajo demuestran que, en algoritmos que trabajan sobre MDT (o en general, modelos bidimensionales o tridimensionales basados en mallas cartesianas) utilizando un barrido radial, la reestructuración de los datos de entrada mediante interpolación de los mismos en la dirección del sector de análisis, un procedimiento cuya complejidad es $O(s \cdot N)$, es una opción altamente recomendable cuando el problema a resolver sea de complejidad superior. El motivo principal no es solo que el alineamiento de datos en memoria reduce los fallos de cache, sino que la preparación de los datos permite que las computaciones de los segmentos paralelos del sector se puedan ejecutar en paralelo.

Además, el número de hilos que es posible crear es proporcional a $\sqrt{N}$, por lo que son claros candidatos para su ejecución en arquitecturas *manycore*.

Como consecuencia de ello, en futuros trabajos se implementará una versión heterogénea, en la que los distintos sectores se ofrezcan a modo de granja de tareas sobre las que CPUs y GPUs seleccionen su trabajo usando un algoritmo de asignación por *work-stealing* o similar.

Agradecimientos

Este trabajo ha sido cofinanciado por el Ministerio de Educación y Formación Profesional, a través del Proyecto TIN2016-80920-R; por la Junta de Andalucía, a través del Proyecto de Excelencia TIC-8260 y por la Universidad de Málaga, a través del I Plan Propio Integral de Docencia y el Proyecto *usmartdrive* del I Programa Smart-Campus.

Referencias

[1] Wm Randolph Franklin and Clark Ray, "Higher isn't necessarily better: Visibility algorithms and experiments," in *Advances in GIS research: sixth international symposium on spatial data handling*. Taylor & Francis Edinburgh, 1994, vol. 2, pp. 751–770.

[2] ESRI, *ARCGIS Software. Version 9.3*, Environmental Systems Research Institute, 2010.

[3] M. Neteler, M.H. Bowman, M. Landa, and M. Metz, "GRASS GIS: a multi-purpose Open Source GIS," *Environmental Modelling & Software*, vol. 31, pp. 124–130, 2012.

[4] Mikhail J Atallah, "Dynamic computational geometry," in *Foundations of Computer Science, 1983., 24th Annual Symposium on*. IEEE, 1983, pp. 92–99.

[5] Leila De Floriani and Paola Magillo, "Visibility algorithms on triangulated digital terrain models," *International Journal of Geographical Information Systems*, vol. 8, no. 1, pp. 13–41, 1994.

[6] Brian Cabral, Nelson Max, and Rebecca Springmeyer, "Bidirectional reflection functions from surface bump maps," in *ACM SIGGRAPH Computer Graphics*. ACM, 1987, pp. 273–281.

[7] David R Miller, Neil A Brooker, and Alistair NR Law, "The calculation of a visibility census for scotland," in *Proceedings of the ESRI annual conference*, 1995.

[8] David Miller, "A method for estimating changes in the visibility of land cover," *Landscape and Urban Planning*, vol. 54, no. 1, pp. 93–106, 2001.

[9] AR Cervilla, S Tabik, EL Zapata, and LF Romero, "Visibilidad total para observadores desligados del terreno," *Jornadas Sarteco*, 2012.

[10] A James Stewart, "Fast horizon computation at all points of a terrain with visibility and shading applications," *Visualization and Computer Graphics, IEEE Transactions on*, vol. 4, no. 1, pp. 82–93, 1998.

[11] S. Tabik, A.R. Cervilla, E.L. Zapata, and L.F. Romero, "Efficient data structure and highly scalable algorithm for total-viewshed computation," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 8, pp. 304–310, 2015.

[12] A.R. Cervilla, S. Tabik, J. Vias, M. Merida, and L.F. Romero, "Total 3d-viewshed map: Quantifying the visible volume in digital elevation models," *Transactions in GIS*, 2016.

[13] A.R. Cervilla, S. Tabik, and L.F. Romero, "Siting multiple observers for maximum coverage: An accurate approach," in *Proceedings of the 2015 International Conference on Computational Science, ICCS2015*, 2015.

[14] A. Cervilla, *Algoritmos Multicore para el Cálculo de Parámetros de Visibilidad en Sistemas de Información Geográfica*, Tesis Doctoral. Universidad de Málaga, 2019.

[15] Siham Tabik, Emilio Zapata, and Luis F. Romero, "Simultaneous computation of total viewshed on large high resolution grids," *International Journal of Geographical Information Science*, vol. 27, no. 4, pp. 804–814, 2013.

On parallelizing Set Inversion by Interval Analysis.

Konstantinos Nasiotis[1], Daniel López[2], Stavros P. Adam [3], Leocadio G. Casado[4]

Resumen— **Engineers use parametrized functions to model observations. Set Inversion via Interval Analysis (SIVIA) characterizes the sets of values of the parameters in the modelling function (usually determined by several equations) for which the values generated are included in the observations. Hence, the fitness of the model can be evaluated. In some cases, the problem comes to determining the values of the parameters for a certain level set of the function. SIVIA is a branch-and-prune algorithm which approximates the sets of parameters' values with unions of axis aligned boxes in the domain of the function, i.e. the search space, based on the characteristic of the images of these boxes to be *inside*, *outside* or *borderline* the sets of observations' values. We are studying an efficient parallel implementation for this algorithm. Our version is based on a dynamic number of threads, at run-time, with inherent dynamic load balancing. A workload estimator for a box and a cutoff of the parallelism are necessary to improve the dynamic load balancing. Here we show our options for them and the speed-up achieved in our implementation.**

Palabras clave— **Level Set, Interval analysis, Parallel branch-and-prune.**

I. Introduction

A. Interval Analysis

Since their emergence, which can be attributed to the work of Moore [1], interval concepts and related numerical techniques have been widely studied from a theoretical point of view resulting in the establishment of the field of Interval Analysis. On the other hand interval methods have been extensively used in several scientific domains for solving both theoretical as well as practical problems in different domains. Among the success stories of interval computations[1] [2] one may report the application of interval based global optimization, validation of numerical calculations, the use of intervals for modelling uncertainty and dealing with uncertain systems, etc.

Hence, interval computations have proven to be extremely important to a number of problems for which errors produced by calculations or due to uncertainty can be modelled in terms of intervals. However, in [3] Kreinovich provides a number of convincing arguments that interval computation are NP-hard and the only way to deal with NP-hardness is to use parallel versions of the algorithms, when this is feasible.

[1]Dept. of Informatics and Telecommunications, University of Ioannina, Greece. e-mail: knasiotis@gmail.com.
[2]Supercomputing Group, University of Almería (CeiA3), Spain. e-mail: dlg697@ual.es.
[3]Dept. of Informatics and Telecommunications, University of Ioannina, Greece. e-mail: adamsp@uoi.gr.
[4]Supercomputing Group, University of Almería (CeiA3), Spain. e-mail: leo@ual.es.

[1]Interval Computations

The following notation is used. A box $[x] \in \mathbb{IR}^n$ is a subset of $\mathbb{R}^n$. It is defined as the Cartesian product of intervals [4]:

$$[x] = [x_1] \times \cdots \times [x_n] = [\underline{x_1}, \overline{x_1}] \times \cdots \times [\underline{x_n}, \overline{x_n}]$$

The width of a box is given by:

$$\omega([x]) = \max_{i=1,\ldots,n} \{\overline{x}_i - \underline{x}_i\}$$

Interval arithmetic is the arithmetic on intervals numbers. If $\diamond$ denotes an interval arithmetic operator, then

$$[x] \diamond [y] = \{x \diamond y \mid x \in [x], y \in [y]\}. \quad (1)$$

For instance, the basic operators on intervals (one dimensional boxes) are:

$$\begin{aligned} [x] + [y] &= [\underline{x} + \underline{y}, \overline{x} + \overline{y}] \\ [x] - [y] &= [\underline{x} - \overline{y}, \overline{x} - \underline{y}] \\ [x] \cdot [y] &= [a, b] \\ a &= \min\{\underline{x}\underline{y}, \overline{x}\underline{y}, \underline{x}\overline{y}, \overline{x}\overline{y}\} \\ b &= \max\{\underline{x}\underline{y}, \overline{x}\underline{y}, \underline{x}\overline{y}, \overline{x}\overline{y}\} \\ \frac{1}{[y]} &= [\frac{1}{\overline{y}}, \frac{1}{\underline{y}}],\ 0 \notin [y] \end{aligned}$$

Substituting real arguments and real operators by their interval counterparts we can obtain the standard interval inclusion function [5]. $[f]([x])$ is an inclusion function of $f(x)$ when $f([x]) \subseteq [f]([x])$. For point values of the arguments, considered as degenerate intervals, $f(x) = [f](x)$. That is not always true when Computational Arithmetic is used due to round-off errors. To maintain the property of inclusion function, round-down (round-up) is set on the computer to calculate a lower (upper) bound of the solution, satisfying $f(x) \subseteq [f](x)$.

Interval arithmetic is up to four times more expensive than Computer Arithmetic, resulting in interval(s) that under and over estimate the real solution(s). The algebraic properties of the Interval Arithmetic are different of the Real Arithmetic: [0,1]-[0,1]=[-1,1] and [1,2]/[1,2]=[1/2,2]. Different expressions of the same function can produce different resulting intervals. Better bounds are obtained when the number of appearance of the variables is reduced. For instance:

- $f_1(x) = x - x^2$. $[f_1]([0,1]) = [-1,1]$.
- $f_2(x) = x(1-x)$. $[f_2]([0,1]) = [0,1]$.
- $f_1([0,1]) = f_2([0,1) = [0,1/4]$.

The relation of a real function f, its interval extension $[f]$ and the inclusion property are illustrated

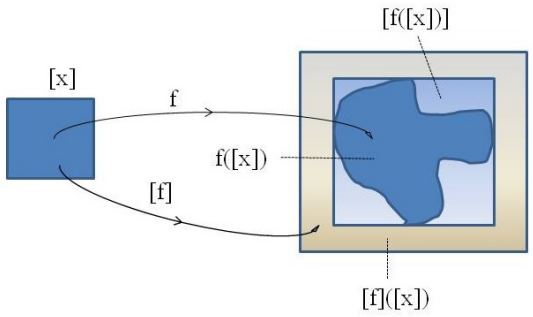


Fig. 1: Inclusion function via Interval Arithmetic.

in Figure 1. One property of the interval inclusion function is the isotinicity. Given $[x] \subseteq [y]$, $[f]([x]) \subseteq [f]([y])$. Standard interval inclusion function has a convergence $\alpha = 1$:

$$\omega([f]([x])) - \omega(f([x])) = \mathcal{O}(\omega([x])^{\alpha}). \qquad (2)$$

So, under and over estimations are reduced as $\omega([x]) \to 0$. Other inclusion functions with a higher convergence order can be found in literature [6].

B. Set Inversion

Set inversion via Interval Analysis (SIVIA) [7] is a technique relying on interval computations and it is also known to suffer from the curse of dimensionality because it performs an exhaustive search by B&B. SIVIA is used for solving problems such as non linear parameter, state or error estimation for systems operating under bounded uncertainty. The method is primarily a set membership technique designed to solve such estimation problems.

Let $\mathbb{X}, \mathbb{Y}$ be a subset of $\mathbb{R}^n$ and $\mathbb{R}^m$ respectively. Given a function $f : \mathbb{R}^n \to \mathbb{R}^m$, the Set Inversion is the characterization of the set:

$$\mathbb{X} = \{x \in \mathbb{R}^n \mid f(x) \in \mathbb{Y}\} = f^{-1}(\mathbb{Y})\}$$

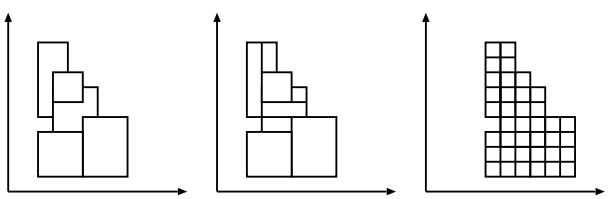


Fig. 2: Left: Image boxes. Center: subpaving. Right Mincing.

In real observations, the set $\mathbb{Y}$ contains overlapped boxes (left graph of Fig. 2). In this study, $\mathbb{Y}$ is a regular subpaving: set of non overlapping boxes with non-zero width (center graph of Fig. 2). An overlapping set of boxes can be converted to a regular subpaving by applying SIVIA to the identity function. Mincing is the division of the subpaving until all boxes are smaller than a given threshold (right graph of Fig. 2). This is done to reduce the under and over estimations of the Interval Arithmetic. In this study, $\mathbb{Y}$ is a subpaving.

II. Related Work

A recent example of SIVIA application is reported by the work of Adam et al [8] who formulated the problem of estimating generalization of a multilayer perceptron as a parameter estimation problem exploring search spaces such as $[-1, 1]^{10}$. Application of a sequential implementation of SIVIA in such experiments revealed extremely interesting results while at the same time it proved the practical impossibility of SIVIA to cope with higher dimensions. This problem is well known to SIVIA practitioners and led to the definition of the so-called contractor programming [9] in order to diminish the size of the boxes explored by SIVIA while some practitioners tried to effectively parallelize SIVIA [10], [11]. As reported in [11], Marvel et al used SIVIA for estimating model parameters in a bounded error context, due to uncertainty in biological systems. They adopted a parallel implementation using multiple processing cores and they developed a method for use on a single multi-core workstation using POSIX threads to process subsets of the parameter space while access to shared information was controlled by mutex-locked linked lists. Four interval box linked lists are used to track parameter sets that are either unclassified, feasible, unfeasible, or indeterminate. In [11] the authors present the results obtained for two biological models, namely, the nonlinear Lotka-Volterra predator-prey model and the SEIR infectious disease model using upto 8 threads on an 8-core machine. Although threads are accessing lists in an exclusive way, super speed-up was reported for some of the execution instances without a reasonable explanation: “due to an increase in L2 cache size when using more than one thread".

Another work that merits to be cited here the computation in real time of the viability kernel as a tool for decision making in autonomous systems [10]. SIVIA was applied to compute the viability kernel of the underlying nonlinear (dynamical) system. [10] shows the results of a parallel implementation of SIVIA on a multi-core processing system while making use of the contractors concept presented above. Due to lack of a detailed description, we deduce that the parallel implementation scheme uses a pool of jobs, each one corresponding to the evaluation of a box. A master-slave scheme is used to distribute the evaluation of boxes. This could be motivated by the used IBEX library which does not allow to share objects between threads, i.e. it is not thread safe. Any performance results were not shown.

III. Sequential SIVIA Algorithm

SIVIA is a Branch and Prune (B&P), special case of Branch and Bound (B&B) algorithm, based on the refinement of the search space until $[f]([x])$ is *in*, *out* of $\mathbb{Y}$ or $\omega([w] \leq \epsilon$ (a *borderline* box). Given an initial set $[x]_0$ which is a full compact set, i.e. a compact sets which is equal to the closure of its interior, and a box $[y]$ enclosing the image of f its is proven that SIVIA converges to the union of boxes $[x]$

whose evaluation in f are in $[y]$ [9]. SIVIA operation may be seen as the following classification of boxes $[x] \in \mathbb{IR}^n$:

In : $[f]([x]) \subseteq [y]$.
Out : $[f]([x]) \cap [y] = \emptyset$.
Borderline : otherwise.

B&B algorithms can be characterized by five rules [12]. For the problem at hand we use the following ones:

1. Division: The largest edge of a box is bisected.
2. Selection: The number of evaluated boxes is independent of the selection method. Depth-First is used because it has the smallest memory requirements.
3. Bounding: Evaluation $[f]([x])$ allows to have an upper and lower bounds of $f(x)$.
4. Elimination: the search does not continue on boxes *in*, *out* of the level set or when $w([x]) \leq \epsilon$.
5. Termination: The algorithm finishes when there not exist more boxes to evaluate.

Algorithm 1 shows the pseudocode of the sequential version of SIVIA. It performs a partition of the search space $[x_0]$ by applying a branch-and-bound (B&B) strategy. Its performance depends on the size of the problem i.e. $\omega([x_0])$, its dimension, the function f, the image set $[y]$ and the achieved precision ϵ.

Algorithm 1 SIVIA(([f], [x_0], [y], ϵ))

Require:
f, ▶ Model function
$[X_0] \in \mathbb{IR}^n$, ▶ Search space
$y \in \mathbb{Y}$, ▶ Image set
ϵ ▶ Elimination criterion
Ensure:
L_{in} ▶ $[f]([x \in L_{in}]) \subseteq [y]$.
L_{out} ▶ $[f]([x \in L_{out}]) \cap [y] = \emptyset$
L_{border} ▶ $\omega([x \in L_{border}]) \leq \epsilon$

$L_{work} \leftarrow [X_0]$ ▶ Working list
while $L_{work} \neq \emptyset$ **do** ▶ Termination criterion
  pop $[x]$ from L_{work} ▶ Selection
  if $[f]([x]) \in [y]$ **then** ▶ Bounding rule
    add $[x] \rightarrow L_{in}$
    continue;
  if $y < [f]([x])$ or $y > [f]([x])$ **then**
    add $[x] \rightarrow L_{out}$ ▶ Elimination
    continue;
  Bisect $[x] = \{[x_1], [x_2]\}$ ▶ Division
  if $\omega([x_1]) < \epsilon$ **then** ▶ Elimination
    add $[x_1] \rightarrow L_{border}$
  else
    push $[x_1] \rightarrow L_{work}$
  if $\omega([x_2]) < \epsilon$ **then** ▶ Elimination
    add $[x_2] \rightarrow L_{border}$
  else
    push $[x_2] \rightarrow L_{work}$
  return L_{in}, L_{out}, L_{border}

SIVIA has been successfully applied in control systems problems with few parameters. For problems with higher dimensions, larger sized input space and fine "resolution" the performance of SIVIA deteriorates severely and becomes practically inapplicable.

IV. Set of test problems

Algorithm 1 has been tested on for problem instances. Following their function formulation is shown.

Example 1: Circle (see Fig. 3a)

$$f(x) = x^2 + y^2,$$
$$[x]_0 = [-1.5, 1.5]^2,$$
$$[y] = [1, 2].$$

Example 2: Griewank (see Fig. 3b)

$$f(x) = \sum_{i=1}^{2} \frac{x_i^2}{4000} - \prod_{i=1}^{2} cos(\frac{x_i}{\sqrt{i}}) + 1,$$
$$[x]_0 = [-10, 10]^2,$$
$$[y] = [1.5, 3].$$

Example 3: Smiley (see Fig. 3c)

$$f(x = \begin{cases} (x^2 + y^2 - 9)(\frac{1}{3}x - y^2) \geq \frac{1}{2} \\ (y-2)^2 + (x-1)^2 \geq \frac{1}{7} \end{cases}$$
$$[x]_0 = [-3, 3]^2,$$
$$[y] = [-\infty, 1/2],\ [-\infty, 1/7].$$

Example 4: Paving from IBEX (see Fig. 3d)

$$f(x) = \sin(x + y) - 0.1xy,$$
$$[x]_0 = [-10, 10]^2,$$
$$[y] = [1.5, 3].$$

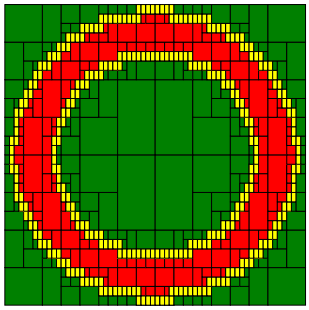
Figure 3 shows a graphical representation of the examples for different ϵ values. Boxes are depicted as: red: *In*, green: *Out* and yellow: *Borderline*.

Table I shows the number of evaluated boxes (and their type) for the ϵ values in Fig.3 and $\epsilon = 10^{-6}$.

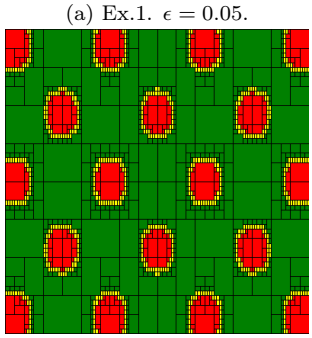
TABLA I: Number of boxes for the examples.

Ex.	Evaluated	In	Out	Borderline
Figure 3				
1	1,303	164	176	624
2	2,959	264	572	1,288
3	2,371	333	287	1,132
4	3,833	467	527	1,846
$\epsilon = 10^{-6}$				
1	94,508,607	13,501,140	13,501,276	40,503,776
2	833,378,959	122,424,640	122,445,084	343,639,512
3	166,073,483	23,122,091	23,141,325	73,546,652
4	1,144,752,563	163,532,496	163,537,411	490,612,750

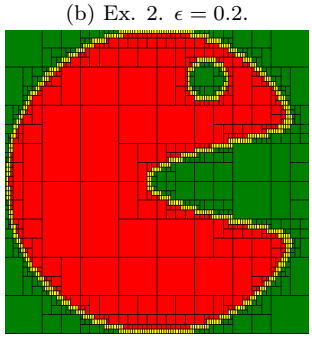
As the accuracy increases the computational burden increases as well, making these problems difficult even for a low dimensional ones. Hence, the need to investigate the possibility of a parallel implementation towards obtaining results in a reasonable time.



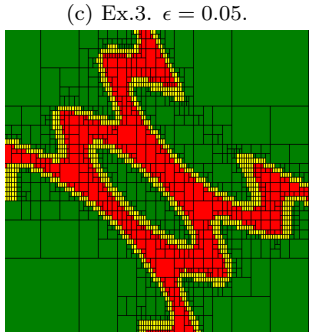
(a) Ex.1. $\epsilon = 0.05$.



(b) Ex. 2. $\epsilon = 0.2$.



(c) Ex.3. $\epsilon = 0.05$.



(d) Ex.4. $\epsilon = 0.2$.

Fig. 3: Graphical representation of the examples.

V. Proposed Parallel Version

Parallelization of B&B codes were widely studied in the literature, see [13] and references therein. In the B&B algorithm the selected box is divided by the widest dimension and Depth-First is used to select next box.

Our parallel implementation is based on a Asynchronous Multiple Pool (AMP) of boxes using a dynamic number of threads [13]. Dynamic load balancing is provided by generating a new thread, if a thread detects that the current number of threads NThreads is smaller than the maximum number Maxthreads which is given as program parameter. The thread that detects the previous condition will select a group of its boxes to reduce its total workload by half and create a new thread to evaluate the selected ones. Each thread works in its own data structures associated to a virtual process unit (VPU). In this way, contention is avoided. The number of VPUs is MaxThreads. When a thread finishes its work it freezes its VPU, decreases NThreads and dies. Several threads can have been associated to a VPU but at a given time there is just one or none threads associated to it. Local final lists in each VPU have to be gathered when all threads end.

We have to define a metric to estimate the workload. The amount of work associated to a box is the size of its search subtree, i.e, the number of subboxes that will be evaluated from it. This is difficult to know beforehand [14]. We use the following heuristic. Given a box $[x]$, it has to be divided more when $[y]$, the image of f, is centred in $[f]([x])$ and $\omega([y]) \ll \omega([f]([x]))$.

We define yInf(f,$[x]$,$[y]$) $\in [0,1]$ depending how $[y]$ is in $[f]([x])$:

- If $\underline{y} > \overline{f}([x])$ or $\overline{y} < \underline{f}([x])$ then yInf=0.
- If $[y] \cap [f]([x]) = [y]$ then yInf=

$$1 - \frac{|(\overline{f}([x]) - \overline{y}) - (\underline{y} - \underline{f}([x]))|}{\omega([f]([x]))} \times 1 - \frac{\omega([y])}{\omega([f]([x]))} \quad (3)$$

- If $[y] \cap [f]([x]) = [z] \subset [y]$ then yInf is calculated using $[z]$ instead of $[y]$ in (3).

The absolute value in Equation (3) says how much is $[y]$ centred in [f]([x]). The multiplier in (3) shows that wider the $[y]$ in relation to $[f]([x])$ lower the workload because subboxes will be *In* in few divisions. When f is defined by several constraints, as in Example 3, yInf is taken as the min of yInf for each constraint because all all them have to be satisfied.

We define the workload of a box $[x]$ as

$$\mathrm{WL}(f, [x], [y]) = \mathrm{yinF}(f, [x], [y]) \times \omega([x]). \quad (4)$$

Therefore, bigger boxes with higher yinF values will have more work than smaller boxes with a small yinF value.

The total workload of a thread is determined by the workload of boxes in its local L_{work} list (see Alg. 1. It is defined as

$$\mathrm{TWL} = \sum_{[x]_i \in L_{work}} \mathrm{WL}(f, [x]_i, [y]) \quad (5)$$

Additionally, a CutOff of parallelism has to be included. It prevent to send to a new thread a box which is worth to evaluate than to migrate.

VI. Experiments

Experiments were run on a node of Bullxual 2 Intel(R) Xeon(TM) E5 2650v2 with 16 cores and 128 Gb of RAM. Interval Arithmetic were provided by C-XSC 2.5.4[2]. Programs were compiled with GNU Gcc and LinuxThreads NPTL.

[2] www.xsc.de

Sequential algorithm uses a FIFO queue for L_{work} with $\mathcal{O}(1)$ for insertion and extraction. Moreover it does not calculate the workload.

Each thread in the parallel version uses an AVL tree to store boxes indexed by their workloads. Each node of the AVL tree has a queue of boxes with the same WL (see (4)). Insertion and extraction in AVL tress is $\mathcal{O}(\log_2 n)$. Depth-First search selects the box with smaller WL from the two generated in a division. When both boxes are final, the box with smaller WL is selected from the AVL tree. AVL tree is used in order to halve the TWL (see (5)) of a thread when it creates another one. Boxes with an appropriate WL that also satisfy the CutOff are searched and extracted from the AVL tree.

Therefore, the management of the data structure for the parallel implementation is more expensive than the one used in the sequential implementation.

In our parallel implementation we established the CutOff as a percentage δ of the workload of the initial box: CutOff=WL$(f, [x]_0, [y]) \times \delta$. We set $\delta = 2^{-2}$ experimentally for all the examples.

Table II shows the differences of the sequential time and the parallel wall clock time with MaxTreads=1. The worklodad evaluation and AVL trees management result in a more wall clock time for the parallel version with one thread than the sequential one.

TABLA II: Wall clock time for sequential and parallel version with one thread. $\epsilon = 10^{-6}$.

Time	Ex. 1	Ex. 2	Ex. 3	Ex. 4
Seq.	135.9	306,3	1575,7	1758,4
1 Th.	304.4	715,8	3545,5	4371,6

Figure 4 shows the speed-up of the parallel algorithm (compared with our best sequential version). The parallel version suffers from parallel overhead (work performed by the parallel algorithm that is not done in the sequential one) even when just one thread is used.

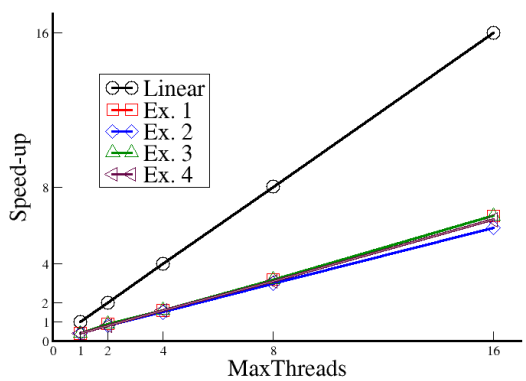


Fig. 4: Speed-up

Figure 5 shows the relative to one thread speed-up of the parallel algorithm. Now the relative speed-up is close to the linear one. This means that the evaluation of the workload and the management of the data structures have a relevant cost in comparison with the useful work (boxes evaluation). Moreover, due to the AMP model previous parallelism overhead is scalable and its cost per core decreases as the number of MaxThreads increases. Therefore, we expect that for larger dimensional problems using a large number of threads/cores the parallel overhead will not be an issue of the parallel algorithm. However, the use of queues in the sequential version will end up exhausting the local memory of each processor slowing down considerably its execution.

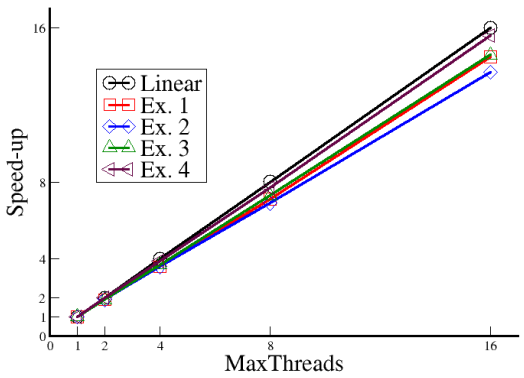


Fig. 5: Relative to one thread speed-up.

Other interesting options have to be checked. For instance, what is the thread in charge of create a new thread and share its workload? The current schema is the first thread noticing NThreads is smaller than MaxTreads. The desirable schema is the thread having more workload. The later incurs in additional parallel overhead because all running threads should know the current workload of the others. Additionally, automatic setting of the CutOff value is a desirable utility of the parallel algorithm.

Moreover, not only the parallel version can be improved but also the sequential one in order to reduce the number of evaluated boxes. For instance, using a better division rule or including contractors.

Acknowlegements

This work has been funded by grants from the Spanish Ministry (RTI2018-095993), in part financed by the European Regional Development Fund (ERDF).

VII. Conclusions

Despite its computational complexity, NP-hard, SIVIA has proved to be a notable method for estimation problems in a bounded uncertainty context. The goal of this ongoing study is to provide an efficient parallel version of SIVIA on a system with a large number of cores. Next versions will include

accelerating techniques and will be able to address larger instances in a reasonable time.

Referencias

[1] Ramon E Moore, *Interval analysis*, vol. 4, Prentice-Hall Englewood Cliffs, NJ, 1966.

[2] Ralph B. Kearfott, "Interval computations: Introduction, uses, and resources," *Euromath Bulletin*, vol. 2, no. 1, pp. 95–112, 1996.

[3] Vladik Kreinovich and Andrew Bernat, "Parallel algorithms for interval computations: An introduction," *Interval Computations*, vol. 1994, 01 1994.

[4] S. Tornil-Sin, V. Puig, and T. Escobet, "Set computations with subpavings in matlab: The scs toolbox," in *2010 IEEE International Symposium on Computer-Aided Control System Design*, Sep. 2010, pp. 1403–1408.

[5] Ramon R. Moore, R. baker kearfott, and Michael J.Cloud, *Introduction to Interval Analysis*, Siam, 2009.

[6] Boglárka Tóth and Tibor Csendes, "Empirical investigation of the convergence speed of inclusion functions in a global optimization context," *Reliable Computing*, vol. 11, no. 4, pp. 253–273, 2005.

[7] Luc Jaulin and Eric Walter, "Set inversion via interval analysis for nonlinear bounded-error estimation," *Automatica*, vol. 29, no. 4, pp. 1053 – 1064, 1993.

[8] Stavros P. Adam, Aristidis C. Likas, and Michael N. Vrahatis, "Evaluating generalization through interval-based neural network inversion," *Neural Computing and Applications*, Mar 2019.

[9] Luc Jaulin, Michel Kieffer, Olivier Didrit, and Eric Walter, "Interval analysis," in *Applied Interval Analysis*, pp. 11–43. Springer, 2001.

[10] Stéphane Le Ménec, "Interval Based Parallel Computing of the Viability Kernel," https://swim2016.sciencesconf.org/data/pages/LeMenec.pdf, 2013, SWIM 2016 – Lyon.

[11] Skylar W. Marvel, Maria A. de Luis Balaguer, and Cranos M. Williams, "Parameter Estimation in Biological Systems Using Interval Methods with Parallel Processing," in *Proc. of the 8th International Workshop on Computational Systems Biology*, Zürich, Switzerland, 2011, pp. 161–168.

[12] L.G. Casado, J.A. Martínez, I. García, and E.M.T. Hendrix, "Branch-and-bound interval global optimization on shared memory multiprocessors," *Optimization Methods and Software*, vol. 23, no. 5, pp. 689–701, 2008.

[13] Juan F. R. Herrera, José M. G. Salmerón, Eligius M. T. Hendrix, Rafael Asenjo, and Leocadio G. Casado, "On parallel branch and bound frameworks for global optimization," *Journal of Global Optimization*, vol. 69, no. 3, pp. 547–560, Nov 2017.

[14] José L. Berenguel, L. G. Casado, I. García, and Eligius M. T. Hendrix, "On estimating workload in interval branch-and-bound global optimization algorithms," *Journal of Global Optimization*, vol. 56, no. 3, pp. 821–844, Jul 2013.

Scheduling paralelo sobre clústeres heterogéneos: Microreología Activa como caso de estudio

G. Ortega[1], F. Orts[2], A.M. Puertas[3], I. García[4], E.M. Garzón[5]

Resumen—**Las plataformas computacionales modernas se caracterizan por la heterogeneidad de sus elementos de procesamiento. Además, hay muchos algoritmos que pueden estructurarse como un conjunto de procedimientos o tareas con distinto coste computacional. El balanceo de la carga computacional entre los elementos de procesamiento disponibles es una de las principales claves para la explotación óptima de tales plataformas heterogéneas. Cuando se conoce el tiempo de procesamiento de cualquier procedimiento ejecutado en cualquiera de los elementos de procesamiento disponibles, tal problema de balanceo de carga de trabajo se puede modelar como el conocido problema de scheduling en máquinas paralelas no relacionadas. Resolver este tipo de problemas es un gran reto debido a la alta heterogeneidad en ambas, las tareas y las máquinas que intervienen. En este artículo, el problema de balanceo se ha definido formalmente como un problema de optimización global que minimiza el makespan (tiempo de ejecución paralelo). Además, una nueva heurística basada en un algoritmo genético, a la que hemos denominado Scheduling Genético (GenS), ha sido desarrollada para resolverlo. Para analizar el comportamiento de GenS en varios clústeres heterogéneos, se ha considerado como caso de estudio un ejemplo tomado del campo de la mecánica estadística: un modelo de microreología activa. Dado este tipo de problema y un clúster heterogéneo, buscamos minimizar el tiempo de ejecución total para extender y analizar en profundidad el caso de estudio. En tal contexto, una tarea consiste en la simulación de una partícula trazadora hallada en una caja cúbica junto con partículas de baño más pequeñas. La carga computacional depende del número total de partículas del baño. Asimismo, GenS se ha comparado con otras estrategias de scheduling dinámicas y estáticas. Los resultados experimentales de la comparación han demostrado que GenS supera al resto de las alternativas logrando un mayor equilibrio de la carga computacional en un clúster heterogéneo. Por lo tanto, la estrategia de scheduling desarrollada en este trabajo es de gran interés para cualquier aplicación que requiera la ejecución de muchas tareas de diferente duración (a priori conocida) en un clúster heterogéneo.**

Palabras clave— **Scheduling paralelo, clúster heterogéneo, máquinas no relacionadas, algoritmo genético.**

[1]Dpt. de Arquitectura de Computadores, Campus de Teatinos, Univ. de Málaga, 29010, Málaga, España, e-mail: gloriaortega@uma.es.

[2]Dpt. de Informática, Univ. de Almería, ceiA3, 04120, Almería, España, e-mail: francisco.orts@ual.es.

[3]Dpt. de Química y Física, Univ. de Almería, ceiA3, 04120, Almería, España, e-mail: apuertas@ual.es.

[4]Dpt. de Arquitectura de Computadores, Campus de Teatinos, Univ. de Málaga, 29010, Málaga, España, e-mail: igarciaf@uma.es.

[5]Dpt. de Informática, Univ. de Almería, ceiA3, 04120, Almería, España, e-mail: gmartin@ual.es.

I. Introducción

Los sistemas computacionales modernos consisten en clústeres heterogéneos compuestos por la interconexión de Unidades de Procesamiento (PUs) con diferentes potencias computacionales, como CPUs, GPUs, etc. [1]. Los algoritmos desarrollados para este tipo de plataformas tienen que tratar tal heterogeneidad para explotar eficientemente los diferentes recursos en las computadoras modernas. A tal efecto, el programador es responsable de seleccionar explícitamente los dispositivos y mapear las tareas entre PUs. Así, las técnicas de scheduling se convierten en uno de los problemas más difíciles, teniendo un tremendo impacto en el rendimiento. Muchos ejemplos de aplicaciones paralelas consisten en un conjunto de tareas independientes, con diferentes costes computacionales, que deben mapearse en un conjunto de unidades de procesamiento heterogéneas de una manera óptima. Este problema puede ser modelado como un scheduling de tareas sobre máquinas paralelas no relacionadas, que es un problema NP-completo [2] y un campo conocido de la computación operativa [3].

Existen dos enfoques para los problemas de scheduling, los dinámicos y los estáticos. Los dinámicos se basan en la definición de una cola global de tareas de la que cada PU disponible elige una nueva tarea. Un scheduling dinámico no necesita ninguna información a priori y, por lo general, es la mejor opción cuando la carga de las tareas es impredecible. Sin embargo, un scheduling dinámico puede producir soluciones que no sean óptimas cuando el tiempo de ejecución de las tareas es muy heterogéneo. En los últimos años han surgido varias interfaces dinámicas para hacer frente a los schedulings en clústeres heterogéneos. Por ejemplo, StarPU [4], Qilin [5] y Scout [6] ofrecen diferentes métodos para asignar tareas tanto a la CPU como a la GPU. La desventaja de estos paradigmas es que requieren que los programadores reescriban sus códigos usando un nuevo lenguaje de programación en el caso de StarPU o Scout o usando API específicas en el caso de Qilin [7].

Por otro lado, los enfoques estáticos son útiles cuando es posible tener una estimación a priori del tiempo de ejecución de las tareas. En tales casos, pueden proporcionar resultados cerca del óptimo ya que consideran el problema desde un punto de vista holístico. Este artículo se centra en tal contexto. Aunque este tipo de problemas es un reto,

puede ser resuelto eficientemente si se considera un conocimiento a priori sobre el tiempo de ejecución de cada tarea en cada una de las máquinas. El scheduling propuesto podría ser de interés para cualquier problema que se plantee que cumpla las premisas anteriormente mencionadas.

En este documento se ha seleccionado como caso de estudio un modelo de Microreología Activa (AMM) en coloides duros para ilustrar el scheduling de simulaciones de sistemas muy grandes en plataformas heterogéneas. Desde el punto de vista computacional, las simulaciones de sistemas muy grandes tienen enormes requisitos y pueden estructurarse como un conjunto de tareas independientes con diferentes cargas computacionales (simulaciones de sistemas con distintos tamaños).

En tales simulaciones de ordenador, uno típicamente intenta evaluar las propiedades de un sistema infinito modelando uno finito. Esto causa indeseables efectos de tamaño finito (FSE), que podrían dominar el comportamiento del sistema [8], enmascarando las propiedades reales de la masa. Existen varias estrategias para luchar contra los FSE, por ejemplo, trabajando con grandes sistemas, utilizando condiciones de límites apropiadas (normalmente condiciones de límites periódicos), o desarrollando un modelo teórico para contabilizarlas. Un ejemplo clásico donde los FSE son muy fuertes es la hidrodinámica, donde las correlaciones decaen muy lentamente, con la distancia inversa [9], [10].

En este artículo, se utiliza un análisis de tamaño finito para extrapolar los resultados de un sistema finito a uno infinito, que requiere simulaciones de sistemas con diferentes tamaños (todos ellos grandes). En la microreología activa, se estudia el comportamiento mecánico y de flujo de un fluido complejo a nivel microscópico [11], [12]. Por lo tanto, para calcular la microviscosidad de un gran sistema, es necesario ejecutar simulaciones de sistemas con diferentes tamaños, y extrapolar al sistema infinito del que depende del modelo. Es importante tener en cuenta que para cada tamaño del sistema, se deben evaluar muchas trayectorias de la partícula trazadora (normalmente 500 en este trabajo) para obtener una buena estimación de la velocidad media de la trazadora. En el contexto de simulaciones en el campo de la AMM, es posible tener una buena estimación a priori del tiempo de simulación en las distintas unidades de procesamiento.

Por lo tanto, un scheduling estático basado en un análisis global es una opción apropiada para optimizar la ejecución paralela de tales simulaciones. En este trabajo, la estrategia de scheduling se define formalmente como un problema de optimización global que minimiza el makespan (tiempo de ejecución paralelo de los procesos de simulación). Luego, se desarrolla una nueva heurística basada en un algoritmo genético para resolver el scheduling en máquinas paralelas no relacionadas. En lo sucesivo, se conoce como Scheduling Genético (Gens). Además, otras estrategias de scheduling (dos dinámicas y dos estáticas) se han revisado y evaluado comparativamente con respecto a GenS. Nuestros resultados han demostrado que GenS supera a los otros métodos de scheduling en términos de makespan y de balanceo de carga utilizando el caso de estudio que hemos comentado anteriormente.

Las principales contribuciones de este artículo pueden resumirse de la siguiente manera: (1) se ha diseñado y evaluado comparativamente una nueva heurística de scheduling basada en un algoritmo genético para distribuir eficazmente las tareas heterogéneas sobre recursos heterogéneos; (2) este scheduling hace factible resolver simulaciones computacionalmente más costosas dentro del campo de la microreología activa; (3) se proporciona un software de scheduling para balancear eficazmente un conjunto de tareas independientes con diferentes costes en unidades de procesamiento heterogéneas, esta estrategia de scheduling se llama GenS, (https:/github.com/2forts/GENS). Por lo tanto, este software puede ser útil para todos los problemas que se puedan modelar mediante un scheduling en máquinas paralelas no relacionadas más allá del caso de estudio que aquí se define.

II. Problemas de Scheduling en máquinas paralelas no relacionadas

Asumamos que un clúster tiene K PUs (Unidades de Procesamiento), esto es, por ejemplo el número total de CPUs más GPUs disponibles. Sea $\{R_m\}$ el conjunto de tareas que define el modelo, con $m = 1, \ldots, M$ y $M = \sum_{i=1}^{I} Q_i$ representa el número total de tareas a computar, I identifica los distintos tamaños N_i, con $1 \leq i \leq I$, y Q_i representa el número de tareas con tamaño de sistema N_i.Luego, el propósito es encontrar un scheduling que minimice el makespan, C_{max}:

$$\begin{aligned} &\text{Encontrar: } X \qquad \text{tal que} \\ &\text{Minimice: } C_{max} \\ &\text{Sujeto a: } t_k = \sum_{i=1}^{I} x_{k,i} t_{k,i} \leq C_{max}, 1 \leq k \leq K \\ &\qquad \sum_{k=1}^{K} x_{k,i} = Q_i, 1 \leq i \leq I \\ &\qquad x_{k,i} \in \{0, 1, \ldots, Q_i\}, 1 \leq k \leq K; 1 \leq i \leq I \end{aligned} \tag{1}$$

donde $x_{k,i}$ representa el número de tareas de tamaño N_i asignadas a la k-th PU; $x_{k,i}$ es un elemento de la matriz, X, que define la asignación de las tareas a las PUs (máquinas); $t_{k,i}$ representa el tiempo de ejecución para computar una tarea de tamaño N_i en la k-th PU. Las restricciones para X significan que cada tarea es computada en una sola PU y cada conjunto de Q_i tareas con el mismo tamaño son distribuidas entre todas las PUs. La fila k-th de X define el conjunto de tareas asignadas a la k-th PU y la i-th columna establece la distribución de las tareas de tamaño N_i entre las K PUs (Ver Fig 1).

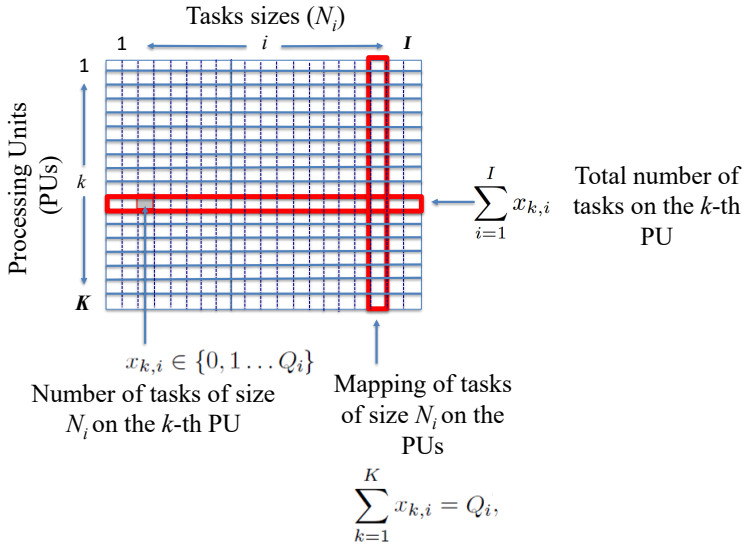


Fig. 1: La matriz X define la asignación de las tareas a las PUs.

Además, se conoce que en los problemas de scheduling C_{max} alcanza el valor mínimo si todas las máquinas completan sus trabajos a la vez, esto es, la carga de trabajo está balanceada entre las máquinas de acuerdo a su potencia computacional [13].

El problema de scheduling definido por la Ec. 1 incluye el tiempo de ejecución de cada tarea en cada PU, $t_{k,i}$. La estimación de $t_{k,i}$ puede ser computada de forma precisa y rápida a priori, ya que $K \times I$ matriz $T = (t_{k,i})$ incluye un alto porcentaje de filas idénticas relacionadas con los mismos tipos de PUs. El tiempo de ejecución de las tareas puede ser caracterizado por una matriz $\mathcal{T}$ de dimensiones reducidas $S \times I$ donde S representa el número de distintos tipos de PUs.

El scheduling sobre máquinas paralelas no relacionadas es un reto debido a la heterogeneidad de ambos, las tareas y la arquitectura clúster. Por tanto, es necesario definir un scheduling de tareas apropiado para obtener el rendimiento paralelo óptimo.

III. Estrategias para el balanceo de la carga

De acuerdo al formalismo introducido, la metodología estática para optimizar la distribución de tareas entre las PUs heterogéneas consta de 3 etapas: (1) Etapa de Profiling, en la que se estima el valor de cada elemento de la matriz T, de acuerdo a los distintos tamaños de los sistemas que están envueltos en el análisis y el número de Unidades de Procesado (PUs); (2) Estimación del scheduling óptimo para identificar el conjunto de tareas que cada PU debe computar, teniendo en mente el conocimiento a priori dado por la etapa de profiling, por tanto, el tiempo paralelo puede también estimarse; y (3) Ejecución paralela de todas las simulaciones en las PUs heterogéneas del clúster de acuerdo al scheduling definido en la segunda etapa.

La estimación del scheduling óptimo (etapa 2) podría ser simplificada si hubiese un sólo tipo de tareas y PUs, ya que nosotros podríamos fácilmente alcanzar una buena solución utilizando una distribución homogénea. Sin embargo, en general, esta estimación es más complicada, incluso si un sólo tipo de PU es considerada, ya que el software paralelo podría incluir tareas con distinta carga computacional. Por supuesto, la complejidad computacional de la estimación del scheduling óptimo crece con altos valores de I, M y K. El scheduling sobre clústeres heterogéneos es NP-completo [14]. Nuestra intención es aplicar una heurística que se acerque a la solución óptima para el problema de scheduling, esta solución consiste en minimizar el makespan/desbalanceo [14].

A. Scheduling Genético (GenS)

Existen trabajos previos donde se utilizan algoritmos genéticos (GAs) para resolver problemas de programación utilizando un esquema estático [15]. En este trabajo se considera un GA para resolver el scheduling de máquinas paralelas no relacionadas sobre clústeres heterogéneos. Un GA trabaja con un conjunto de individuos que representan las posibles soluciones del problema: (*population*). Es un procedimiento iterativo que comienza con un conjunto aleatorio de individuos, $P(0)$, y en cada iteración, $iter$, se aplican a la población tanto un mecanismo de selección como operadores genéticos, $P(iter)$. Por lo tanto, la población está en constante evolución. El mecanismo de selección permite a los individuos de la siguiente generación estar más cerca de la solución óptima (ver Algoritmo 1).

Para aplicar el Algoritmo 1 al problema de encontrar un scheduling cercano al óptimo, es necesario especificar los siguientes conceptos: individuo, función de ajuste (fitness), y operadores genéticos (cruzamiento y mutación). Proponemos adaptar un GA al problema de scheduling mencionado anteriormente, obteniendo como resultado el algoritmo GenS.

Teniendo en cuenta el formalismo introducido an-

Algorithm 1 Algoritmo Genético

1: $iter \leftarrow 0$
2: Inicializar Población aleatoriamente $P(0)$
3: Evaluar la función de aptitud (fitness) para la población $P(0)$
4: **while** Condición de parada no es cierta **do**
5: $iter \leftarrow iter + 1$
6: Seleccionar $P(iter)$ de $P(iter - 1)$
7: Aplicar operadores genéticos (cruzamiento y mutación) a $P(iter)$
8: Evaluar la función fitness para $P(iter)$
9: **end**

teriormente, cada individuo en $P(iter)$ está representado por una matriz $K \times I$, llamada X, que define la asignación de tareas a las PUs de acuerdo con la definición de la Ec. 1 (y Fig 1). Después de la evaluación de la función fitness (UB) para toda la población, se ordenan los individuos según su fitness. Así, los individuos con UB más pequeños serán seleccionados a medida que el GA avanza.

En cada iteración del Algoritmo 1, se aplican dos operaciones a la población para promover la evolución. En primer lugar, se define un conjunto aleatorio de pares de individuos (padres) y luego, los nuevos individuos (hijos) son producidos por el operador de cruzamiento (recombinación o crossover). Por tanto, se aplica la conocida recombinación en un punto. La Fig 2 describe cómo se aplica el crossover. Se selecciona una columna aleatoria para dividir las matrices de ambos padres y los nuevos individuos se generan intercambiando los cuatro conjuntos de columnas. Con este esquema, los hijos pueden ser considerados como soluciones válidas ya que se verifican las restricciones para las columnas de sus matrices. Después del crossover, el operador de mutación actúa sobre cada descendiente y puede alterar la distribución de cada columna (con una probabilidad de un 1%). Es un intercambio aleatorio de tareas del mismo tamaño entre un par de PUs, i. e. elementos que estén en la misma columna de la matriz correspondiente podrían intercambiar parcialmente sus tareas. Cada iteración comienza con el mismo tamaño de población (PS). La fase de selección sólo consiste en elegir/ mantener los mejores PS individuos ya que la población se ha ordenado previamente de acuerdo a la función aptitud, UB. El procedimiento se detiene cuando la UB en las 10 últimas iteraciones no ha cambiado para el 30 de los mejores individuos. Resumiendo, si $\{t_{k,i}\}$ con $1 \leq k \leq K$ y $1 \leq i \leq I$ es conocido, GenS es capaz de identificar una distribución cerca a la óptima de las tareas entre el conjunto de PUs.

En general, se desconoce la exactitud de las soluciones obtenidas por los Algoritmos Genéticos. Sin embargo, si UB es cercano a cero, podemos concluir que el individuo correspondiente es casi un óptimo.

B. Estrategias estáticas adicionales

Un ejemplo de enfoque estático es el esquema de aproximación de tiempo polinomial (PTAS) [14]. El algoritmo PTAS puede dar una buena estimación del scheduling óptimo, con la ventaja adicional de que es posible estimar teóricamente la relación con la solución óptima. Sin embargo, el PTAS tiene una elevada sobrecarga computacional debido a la gran cantidad de información que es necesaria almacenar. Esta es la razón por la que el PTAS no ha sido incluido en el estudio comparativo con GenS.

Un enfoque alternativo a GenS consiste en una distribución cíclica de las tareas sobre el conjunto de PUs. En primer lugar, las tareas se ordenan de acuerdo a su carga computacional. Posteriormente, se distribuyen en un orden cíclico entre las PUs. De esta manera, cada PU calcula porcentajes similares de tareas de diferentes costes. De aquí en adelante, este esquema se referirá como Cíclico. Es probable que consiga un scheduling casi óptimo en un clúster homogéneo.

Una heurística Voraz siguiendo el esquema considerado en [16] también se puede definir para resolver el problema de Scheduling. Para un determinado tamaño del sistema, (N_i), la estimación a priori del tiempo de ejecución nos permite identificar la PU más lenta, y el factor de aceleración del resto de PUs con respecto a ella. Estos factores definen el porcentaje de la tarea que se ejecutará en cada PU. Por lo tanto, cada conjunto de Q_i simulaciones para el mismo tamaño del sistema se distribuyen. Los PUs con menor potencia computacional calculan pocas simulaciones y las PUs con la mejor potencia calcularán el porcentaje de tareas definidas por el correspondiente factor de aceleración. Este procedimiento se repite para cada subconjunto de tareas, obteniendo una distribución cercana a la óptima en cada caso, con el objetivo de obtener una solución global óptima.

C. Estrategias Dinámicas

Pueden definirse varios tipos de políticas de scheduling sin una estimación a priori del tiempo de ejecución de las tareas. Sin embargo, nuestro interés se centra en dos enfoques dinámicos que utilizan parcialmente esta información, ya que el punto de partida de ambos es una cola de tareas ordenadas de acuerdo a su carga computacional, N_i.

La Cola de Tareas Sencillas (STQ) resuelve el problema dinámicamente, manejando una sola cola. Cada PU calculará una tarea desde esta cola y cuando termine, tomará una nueva tarea de la cabeza de la cola.

Otro enfoque dinámico es el uso de una Cola Doblemente Terminada en combinación con una clasificación de los dispositivos en dos categorías, dispositivos lentos y rápidos. En este esquema los dispositivos más lentos toman las tareas más ligeras de la cola, y los dispositivos más rápidos las más pesadas. Este scheduling se denominará DETQ y, además, se considerará una información a priori so-

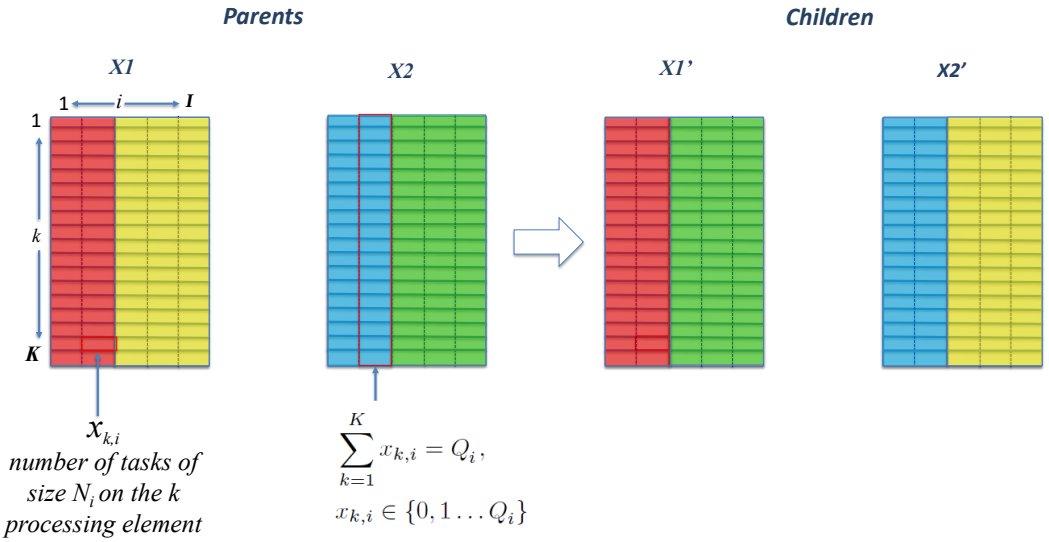


Fig. 2: Procedimiento de cruzamiento (Crossover) para crear nuevos individuos de la población.

bre las cargas de tareas y la potencia de las máquinas.

IV. Modelo de Microreología Activa (AMM) como caso de estudio

Como se mencionó anteriormente, AMM se considera aquí como el caso de estudio porque desde un punto de vista computacional se puede ver como un conjunto de tareas independientes de diferentes cargas que se pueden ejecutar en clústeres heterogéneos. En la microreología activa, se estudia el comportamiento y la mecánica del flujo de un fluido complejo a nivel microscópico [17], [12]. Para ello, una partícula intrusa, típicamente de tamaño coloidal, es introducida y golpeada en un sistema, y su dinámica se monitoriza. En particular, la microviscosidad puede calcularse a partir de la velocidad estacionaria de dicha partícula trazadora a lo largo del tiempo.

En nuestro caso de estudio, el fluido huésped se modela como esferas Brownianas casi-duras, imitando coloides duros. El movimiento Browniano es descrito por la ecuación de Langevin [18], donde la partícula j lee:

$$m_j \frac{d^2\vec{r}_j}{dt^2} = \sum_i \vec{F}_{ij} - \gamma_j \frac{d\vec{r}_j}{dt} + \vec{\eta}_j(t) \quad + \vec{F}_{\text{ext}}\delta_{j1} \tag{2}$$

donde los términos en el r.h.s. son las fuerzas de interacción ($\sum_i \vec{F}_{ij}$), la fricción con el solvente ($-\gamma_j \frac{d\vec{r}_j}{dt}$), la fuerza aleatoria ($\vec{\eta}_j(t)$), y la fuerza externa ($\vec{F}_{\text{ext}}\delta_{j1}$), respectivamente; γ_j es el coeficiente de fricción con el disolvente, que se relaciona con la fuerza aleatoria a través del teorema de disipación de la fluctuación [18], y depende linealmente del radio de partículas. La fuerza externa, $\vec{F}_{\text{ext}}$, que actúa sólo sobre la trazadora, etiquetado por $j = 1$, es constante en nuestro modelo (este hecho es expresado por el bien conocido delta de Kronecker, denotado por δ_{j1}). Las fuerzas de interacción se derivan del potencial entre partículas $V(r_{ij}) = k_{\text{B}}T(r_{ij}/d_{ij})^{-36}$, donde r_{ij} es la distancia centro a centro, y $d_{ij} = (a_i + a_j)/2$, donde a_i es el radio de la partícula i.

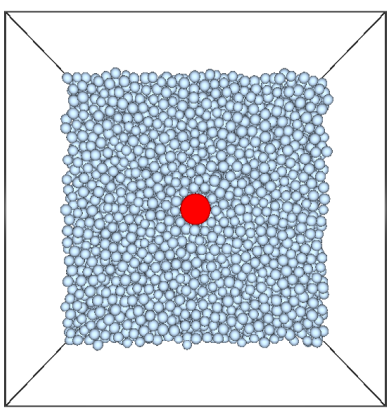


Fig. 3: Captura de un sistema con $N = 15625$ partículas. La trazadora, 3 veces mayor que el baño de partículas, $a_t = 3a_b$, está marcada en color rojo , y las partículas de delante de ella han sido borradas.

Las simulaciones se ejecutan en una caja cúbica, con N partículas y condiciones de frontera periódicos. Las partículas del baño y de la trazadora tienen radios a_b y a_t, respectivamente, y todas las partículas tienen la misma masa, m. En nuestras simulaciones, la unidad de longitud es a_b, la unidad de masa es m, y la unidad de energía es la energía térmica $k_{\text{B}}T$, donde k_{B} es la constante Boltzmann, dando una unidad de tiempo de $a_b\sqrt{m/k_{\text{B}}T}$. La fracción de volumen del fluido se establece en $\phi = 0.50$, el coeficiente de fricción del solvente de la partícula i se establece a $\gamma_0 = 5a_i\sqrt{mk_{\text{B}}T}/a_b$, y $a_t = 3a_b$. Las ecuaciones de movimiento se integran con un paso de tiempo de $0.0005a_b\sqrt{m/k_{\text{B}}T}$ usando una extensión de la velocidad del algoritmo de Verlet para incluir fuerzas aleatorias [19], integrando las fuerzas de fricción analíticamente. La fuerza externa se establece a $F_{ext} = 2.5{,}k_{\text{B}}T$. La Fig 3 muestra una captura de un sistema con N=15625 partículas.

En las simulaciones, la partícula trazadora se em-

puja con una fuerza constante, y su trayectoria es registrada. El coeficiente de fricción efectivo de la trazadora con el baño, γ_{eff}, se obtiene de la velocidad promedio de la trazadora usando la relación de estado estacionaria: $\vec{F}_{\text{ext}} = \gamma_{\text{eff}}\langle\vec{v}\rangle$. Por lo tanto, se necesita un gran número de trayectorias para obtener valores fiables de γ_{eff}. Sin embargo, la trazadora distorsiona el baño a medida que se desplaza, y como es mucho más grande que las partículas del baño, los FSE pueden estar presentes. De hecho, debido a las condiciones de frontera periódicas, un array de partículas se simula con una cuadrícula igual al tamaño de la caja. A partir de la ecuación de Navier-Stokes, Hasimoto [20] demostró que el coeficiente de fricción efectivo medido por un array de partículas en un incomprensible fluido Newtoniano depende de tal cuadrícula, L, así:

$$\frac{1}{\gamma_{\text{eff}}} = \frac{c}{L} + \frac{1}{\gamma_\infty} \tag{3}$$

donde c es una constante que depende de la estructura del array, y γ_∞ es el coeficiente de fricción efectivo medido por una partícula aislada [20]. Siguiendo este resultado teórico, γ_∞ se puede obtener ejecutando simulaciones con diferentes tamaños de sistema, L, para obtener $\gamma_{\text{eff}}(L)$, y extrapolar linealmente a $1/L \to 0$. Es importante tener en cuenta que cambiar el tamaño del sistema implica cambiar el número de partículas porque la fracción de volumen es constante.

La Fig 4 muestra los resultados de γ_{eff} para siete tamaños de sistema, con el número de partículas que van desde $N = 216$ a 15625. El coeficiente de fricción inverso es de hecho lineal para sistemas pequeños, pero se desvía para $1/L \to 0$, debido a las aproximaciones en el modelo teórico.

El análisis completo de los efectos de tamaño finito en el sistema requiere un gran número de simulaciones o tareas de *i*) sistemas con distintos números de partículas, N_i con $1 \leq i \leq I$, y *ii*) un gran número de trayectorias (Q_i) para cada tamaño del sistema (N_i), requiriendo *iii*) resolver N_i ecuaciones de movimiento repetidamente para cada trayectoria. Por lo tanto, los requisitos computacionales de los modelos AMM son muy altos, los cuales estarían proporcionados por los clústeres modernos multi-GPU. El modelo presenta varios niveles de paralelismo, lo que permite la explotación adecuada por parte de tales clústeres. Los trabajos anteriores se centraron en acelerar el cómputo de una sola trayectoria de la trazadora (nivel de paralelismo más bajo) sobre una GPU [21], [22]. Sin embargo, para avanzar en este tipo de modelos, era necesario ejecutar eficientemente muchas trayectorias de forma paralela sobre clústeres heterogéneos (el nivel más alto de paralelismo).

De esta manera, el modelo define un conjunto de $M = \sum_{i=1}^{I} Q_i$ tareas que computan cada trayectoria de la trazadora. Así, las trayectorias de la trazadora se pueden calcular en paralelo en los núcleos CPU y GPU de un clúster. Cada CPU-core (GPU) puede ejecutar el código secuencial (código CUDA) para computar una trayectoria de la trazadora, y el conjunto de tareas puede calcularse con la colaboración de todas las unidades de procesamiento (PUs) del clúster, CPU-cores y GPUs (PUs). Por lo tanto, para obtener una explotación óptima de los clústeres heterogéneos ejecutando modelos AMM es necesario resolver el problema de scheduling en máquinas paralelas no relacionadas definidas en la Sec. II.

V. Resultados

En esta sección, las estrategias mencionadas anteriormente (GenS, Cíclica, Voraz, STQ y DETQ) son evaluadas en una amplia variedad de clústeres heterogéneos. Todas las pruebas se han realizado usando el mismo problema: tamaños de sistema de $N_i = 216$, 512, 1000, 2197, 4096, 8000 y 15625, con 250 trayectorias de 500 unidades temporales (correspondientes a pasos temporales de 10^6). Se han considerado cuatro tipos de PU:

$Core_1$: 1 core de Bullx R424-E3 Intel Xeon E5 2650 con 8GB RAM
GPU_1 : NVIDIA Tesla M2070 GPUs (Fermi)
$Core_2$: 1 core de Bullx R421-E4 Intel Xeon E5 2620v2 con 64GB RAM
GPU_2 : NVIDIA Kepler GK210 (NVIDIA K80)

Las características de las GPU se muestran en la Tabla I. A partir de las PU mencionadas, se han definido siete pruebas (cinco clústeres heterogéneos y dos homogéneos) para evaluar los diferentes métodos (ver Tabla II).

Se han considerado dos implementaciones para simular cada trayectoria de la trazadora: una versión CPU secuencial implementada en Fortran y una versión para GPU implementada en ANSI C y CUDA. Además, se ha utilizado un módulo de multiprocesamiento de Python para codificar los esquemas de procesamiento conseguidos en la evaluación experimental.

TABLA I: Características de las GPU.

	M2070 (GPU_1)	GK210 (GPU_2)
Rendimiento máximo (double prec.) (TFlops)	0.51	2.91
Rendimiento máximo (simple prec.) (TFlops)	1.03	8.74
Memoria del dispositivo (GB)	5.2	24
Velocidad de reloj (GHz)	1.2	0.82
Ancho banda mem. (GBytes/s)	150	480
Multiprocesadores	14	13
nucleos CUDA	448	2496
Capacidad de computo	2.0	3.7

TABLA II: PUs incluidos en cada clúster (A–F).

	$Core_1$	GPU_1	$Core_2$	GPU_2	K
A	14	2			16
B	28	4			32
C	28	8			36
D	56	8			64
E	56	8	10	2	76
F		8			8
G	64				64

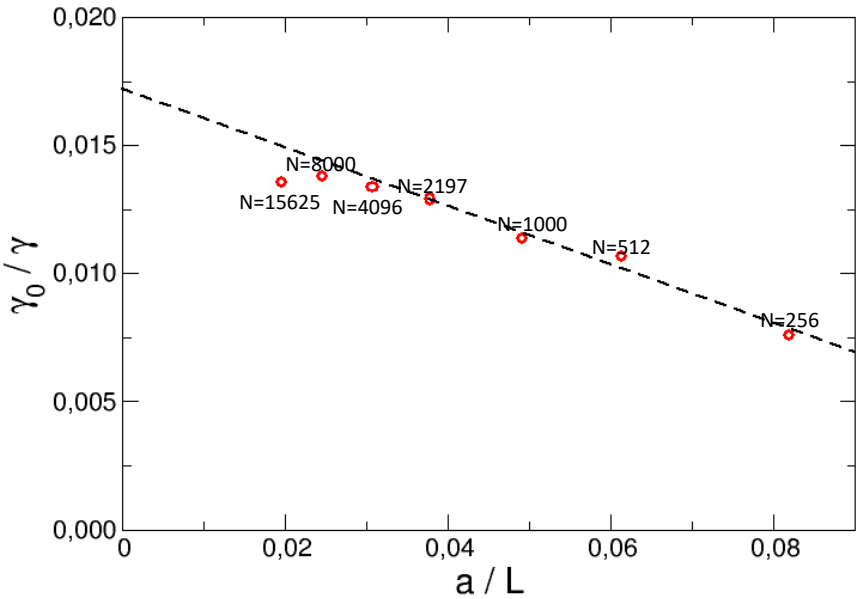


Fig. 4: Inverso del coeficiente de fricción vs. inverso de la longitud de la caja de simulación para un sistema con una fracción de volumen de $\phi = 0.50$, y una trazadora de tamaño $a_t = 3a_b$ empujada con una fuerza $F = 2.5\, k_{\mathrm{B}}T/a_b$. Las etiquetas indican el número de partículas utilizadas en cada simulación. El número de trayectorias analizadas para cada punto es de 500. Recuerde que γ_0 es el coeficiente de fricción del disolvente y γ_∞ se puede obtener de la interceptación de of $\gamma_0/\gamma_{\mathrm{eff}}$ con $\frac{a_b}{L} = 0$.

En primer lugar, se ha llevado a cabo una primera etapa de preprocesado. En esta etapa se obtuvieron todos los tiempos para cada tipo de tarea en cada tipo de dispositivo (matriz $\mathcal{T}$), tal y como se muestra en la Tabla III. AF es el factor de aceleración de cada tipo de dispositivo comparado con el dispositivo más lento para cada tamaño de sistema. Estos valores se utilizan en la estrategia Voraz, como se mencionó en la sección anterior. Es importante resaltar que el tiempo de ejecución aumenta conforme lo hace N_i. Además, el uso de las GPUs no resulta beneficioso para acelerar los problemas de microrreología para los tamaños de sistema más pequeños. Sin embargo, cuando $N_i \geq 1000$, las GPU comienzan a resultar provechosas.

En segundo lugar, centrando la atención en la segunda etapa de nuestra metodología, la Tabla IV muestra el tiempo de ejecución estimado en horas (C_{max}) en los diferentes clústeres (en horas). Al analizar las plataformas homogéneas (F y G), se observa que GenS consigue el mejor rendimiento, igualando o mejorando a las otras estrategias en un 3%. Por lo tanto, para estas plataformas, las ventajas de GenS se mantienen aunque no son muy relevantes. Pero en las plataformas heterogéneas (A - E), los resultados de GenS son significativamente mejores, y su rendimiento aumenta a medida que aumentan la heterogeneidad del problema y el tamaño del clúster. La estrategia Cíclica siempre tiene el peor tiempo de ejecución, STQ obtiene un tiempo de ejecución razonable cercano al DETQ (el segundo mejor), y la Voraz, aunque mejora a la cíclica, sigue teniendo un tiempo elevado en comparación al resto.

Es importante destacar que debido al comportamiento no determinista de GenS, este se ha ejecutado 10 veces para verificar su robustez. La dispersión de los resultados ha sido inferior a $0,15\%$, por lo que podemos destacar su gran robustez.

Para demostrar que una búsqueda aleatoria no es competitiva con respecto a GenS, se ha ejecutado durante un intervalo de tiempo significativamente mayor que el tiempo de ejecución de GenS para resolver el mismo problema. Los resultados han demostrado que GenS supera a la búsqueda aleatoria en términos de makespan. Por motivos de claridad del artículo, esta prueba no ha sido incluida en él.

Centrándonos en el clúster heterogéneo más grande, E. Las figuras reffigura:balanceo(a-e) muestran el tiempo de ejecución de cada dispositivo para cada estrategia y (f) muestra el porcentaje de tareas de cada tamapo ejecutadas en cada plataforma utilizando la solución dada por GenS (los colores muestran el tamaño de la tarea, como se indica en la leyenda). En la estrategia STQ (a), las tareas grandes ,que consumen mucho tiempo, se calculan en los cores de las CPUs, mientras que las GPUs están inactivas, lo que provoca importantes desbalanceos. En la estrategia DETQ (b), el número de tareas grandes que llegan a los cores no es tan importante (el $Core_2$ no toma ninguna, por ejemplo) y, por lo tanto, se reduce el desequilibrio, pero todavía hay grandes desbalanceos. La estrategia cíclica (d) realiza la distribución sin tener en cuenta la heterogeneidad de la plataforma, por lo que es claramente la peor. La estrategia voraz también está lejos de ser óptima. GenS intenta ajustarse a la heterogeneidad de las tareas y al hardware de la plataforma, por lo que su evolución hace que las PU más potentes

TABLA III: Tiempo de ejecución total (en segundos) para siete tamaños de sistema (N_i). Las columas $t_{GPU1,i}$ ($t_{GPU2,i}$) y $t_{CPU1,i}$ ($t_{CPU2,i}$) identifican el runtime para computar una sola trayectoria en una GPU de tipo 1 (2) y en un core-CPU de tipo 1 (2). AF_s, con $1 \leq s \leq 4$, son los factores de aceleración de cada tipo de dispositivo en contraste al dispositivo más lento para cada valor de N_i.

	$s=1$	$s=2$	$s=3$	$s=4$				
N_i	$t_{GPU1,i}$	$t_{CPU1,i}$	$t_{GPU2,i}$	$t_{CPU2,i}$	AF_1	AF_2	AF_3	AF_4
216	1580	790	1406	101	1,0	2,0	1,1	15,6
512	1785	1860	1714	507	1,0	1,0	1,1	3,7
1000	2240	3715	2030	2319	1,7	1,0	1,8	1,6
2197	2930	8710	2112	5315	3,0	1,0	4,1	1,6
4096	4450	18065	3235	10465	4,1	1,0	5,6	1,7
8000	7650	43080	4587	24427	5,6	1,0	9,4	1,8
15625	12050	113940	10043	63788	9,5	1,0	11,3	1,8

TABLA IV: Tiempo de ejecución estimado, en horas, para cada estrategia para los casos especificados en la Tabla II. Los mejores casos están marcados en negrita.

	Heterogéneos					Homogéneos	
	A	B	C	D	E	F	G
STQ	489,6	244,8	184,8	129,6	108,0	**283,2**	211,2
DETQ	412,8	223,2	141,6	122,4	103,2	**283,2**	211,2
Voraz	504,0	261,6	177,6	136,8	112,8	290,4	211,2
Cíclico	844,8	422,4	369,6	211,2	211,2	290,4	211,2
GenS	**410,4**	**206,4**	**139,2**	**102,5**	**79,2**	**283,2**	**206,4**

TABLA V: Estimación (etapa 2) and resultados reales (etapa 3), en horas, obtenidos por GenS, para los clústeres heterogéneos D y E de la Table II.

	Estimación		Reales	
	D	E	D	E
C_{Max}	102,5	79,2	105,4	82,5
C_{Min}	101,8	76,4	104,4	79,4

calculen las tareas más grandes (por ejemplo, $Core_2$ tiene más tareas grandes que $Core_1$) y los dispositivos menos potentes se ocupan de las tareas más pequeñas (Fig 5(f)). Si analizamos el desbalanceo entre las diferentes plataformas en la Fig 5(e), podemos concluir que la solución que aporta GenS no está lejos de ser la óptima, ya que todos los dispositivos terminan casi al mismo tiempo.

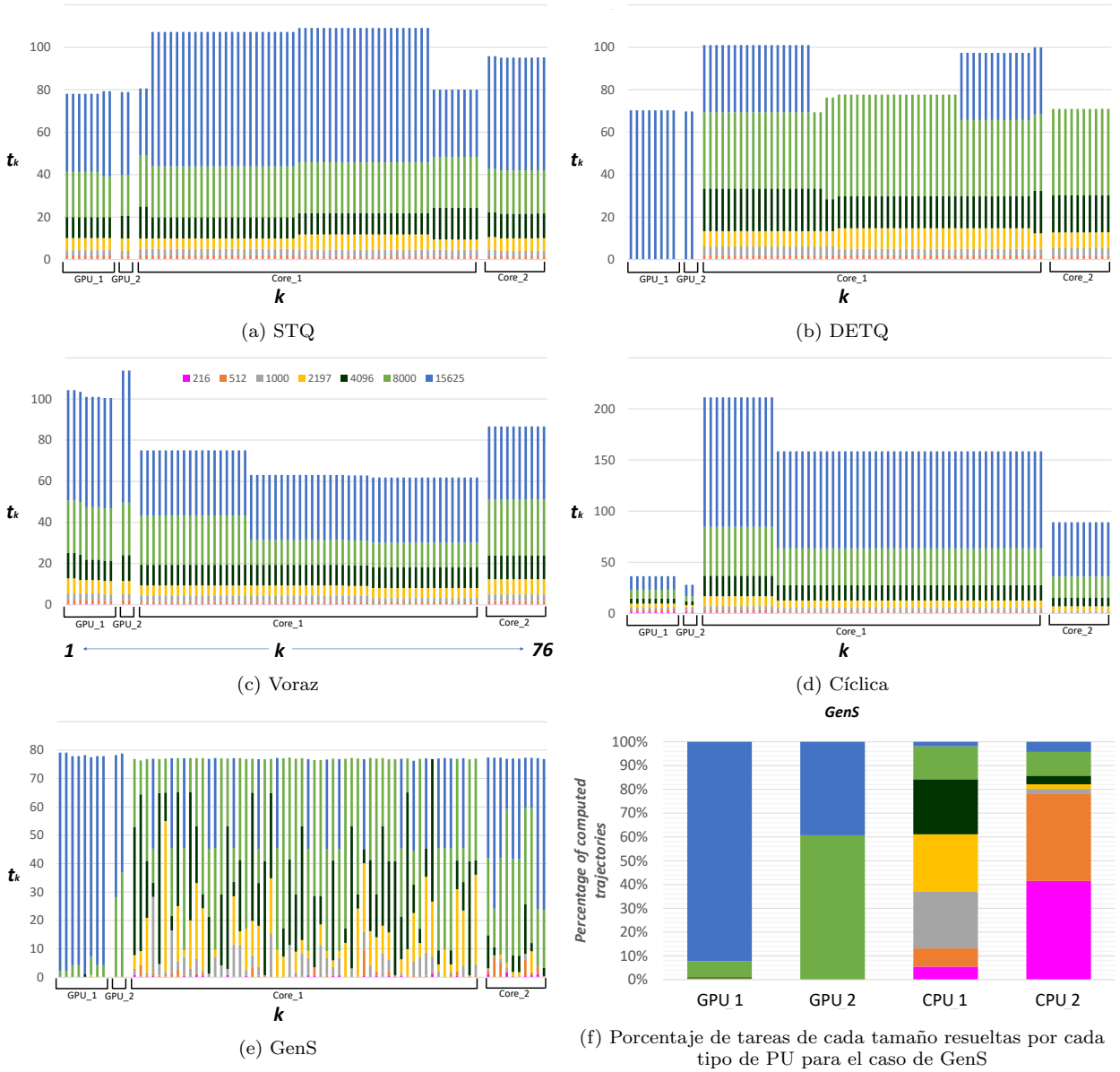
Por lo tanto, GenS obtiene las mejores estimaciones. Una vez hechas las predicciones, el siguiente paso es realizar pruebas reales (etapa 3) para verificar que las estimaciones de GenS sean realistas. Se ha probado ejecuciones reales en los clústeres D y E (los clústeres más grandes y heterogéneos). La tabla VI muestra el tiempo estimado por GenS, en horas, en comparación con las ejecuciones reales siguiendo las planificaciones hechas por GenS. Ya que ambas pruebas dan tiempos muy similares, se considera innecesario realizar más repeticiones. Analizando los tiempos de ejecución, se puede concluir que la estimación de GenS está muy cerca de los resultados reales. El tiempo de ejecución experimental es un poco más grande que el previsto debido a que la estimación de GenS no tiene en cuenta el tiempo necesario para preparar y enviar una tarea a la máquina correspondiente.

Así pues, el uso de GenS para programar el orden de ejecución de las tareas para un modelo de microreología en un clúster heterogéneo ha dado como resultado una importante reducción del tiempo de ejecución, que hasta ahora ha sido un obstaculo en la obtención de nuevos resultados en ese área. Por ejemplo, si el conjunto de simulaciones se calcula en $Core_1$, el tiempo de ejecución secuencial estimado sería de $13205,6$ horas. Por lo tanto, se logra un factor de aceleración de $\times 129$ ($\times 167$) en el clúster D (en el clúster E) utilizando la programación dada por GenS.

VI. Conclusiones

En este trabajo, se ha estudiado la forma de ejecutar un conjunto elevado de tareas de diversos tipos en plataformas heterogéneas. Se ha analizado una solución basada en un algoritmo genético (GenS) con el objetivo de distribuir la carga de trabajo de forma casi óptima. GenS ha sido evaluado en comparación a otras estrategias, y se ha considerado un caso de estudio basado en un Modelo de microreorología activa. El objetivo de tales modelos es el cálculo del coeficiente de fricción efectivo de fluidos complejos donde predominan los efectos de tamaño finito. El coste computacional de estos modelos es enorme ya que se basan en el análisis estadístico de la dinámica de una partícula trazadora para varios tamaños de sistemas. Por lo tanto, la distribución adecuada de las tareas en clústeres heterogéneos ha sido clave para extender la aplicabilidad de estos modelos.

Los resultados han demostrado que GenS logra un equilibrio de carga casi óptimo, incluso cuando el clúster incluye un conjunto grande y heterogéneo de dispositivos, superando al resto de estrategias estudiadas. GenS mejora el rendimiento con respecto a la segunda estrategia más rápida (DETQ) hasta en



(a) STQ

(b) DETQ

(c) Voraz

(d) Cíclica

(e) GenS

(f) Porcentaje de tareas de cada tamaño resueltas por cada tipo de PU para el caso de GenS

Fig. 5: Las imágenes (a–e) muestran el tiempo de ejecución, en horas, para todos los dispositivos de cada estratigia en el clúster E. Cada columna k, mostradas como barras apiladas, corresponde a un dispositivo. Una barra apilada representa el tiempo empleado por el dispositivo k para computar la tarea de tamaño i asignada al dispositivo, por lo que la columna tota representa su tiempo total de ejecución. La imagen (f) muestra el porcentaje de tareas de cada tamaño asignadas por GenS a cada tipo de dispositivo.

un 23, 56% en el clúster E (el más heterogéneo). Por lo tanto, las ventajas de GenS son más relevantes a medida que aumenta la heterogeneidad del clúster.

Solo la evolución de GenS ha permitido definir la asignación de tareas / unidad de procesamiento de acuerdo con la carga de tarea / potencia computacional de forma óptima. De esta manera, todos los dispositivos involucrados terminan su cómputo casi simultáneamente. Una definición adecuada de los operadores e individuos involucrados en el Algoritmo Genético ha sido fundamental para lograr estos resultados.

La principal contribución de este trabajo ha sido proporcionar un software para distribuir de manera eficiente un conjunto de tareas independientes con diferentes costes en unidades de procesamiento heterogéneas (https://github.com/2forts/GENS). Por lo tanto, este software puede ser útil para todos los problemas en los que sea necesario distribuir un conjunto elevado de tareas de diversos tipos en un conjunto de dispositivos también heterogéneo.

Agradecimientos

Este trabajo ha sido parcialmente financiado por el Ministerio de Ciencia de España a través del Proyecto RTI2018-095993-B-I00, por la Junta de Andalucía a través del Proyecto P12-TIC301 y por el Fondo Europeo de Desarrollo Regional (FEDER). F. Orts es becario FPI (Proyecto TIN2015-66680-C2-1-R) del Ministerio de Educación de España. G. Ortega es becaria del programa nacional “Juan de la Cierva-Incorporación”

Referencias

[1] J. Hennessy and D. Patterson, *Computer architecture: a quantitative approach*, Elsevier, 2017.

[2] JK. Lenstra, DB. Shmoys, and E. Tardos, "Approximation algorithms for scheduling unrelated parallel machines," *Mathematical programming*, vol. 46, no. 1-3, pp. 259–271, 1990.

[3] D.B. Shmoys and E. Tardos, "An approximation algorithm for the generalized assignment problem," *Mathematical programming*, vol. 62, no. 1-3, pp. 461–474, 1993.

[4] C. Augonnet, S. Thibault, R. Namyst, and PA. Wacrenier, "Starpu: a unified platform for task scheduling on heterogeneous multicore architectures," *Concurrency and Computation: Practice and Experience*, vol. 23, no. 2, pp. 187–198, 2011.

[5] C.K. Luk, S. Hong, and H. Kim, "Qilin: exploiting parallelism on heterogeneous multiprocessors with adaptive mapping," in *Microarchitecture, 2009. MICRO-42. 42nd Annual IEEE/ACM International Symposium on*. IEEE, 2009, pp. 45–55.

[6] P. McCormick, J. Inman, J. Ahrens, J. Mohd-Yusof, G. Roth, and S. Cummins, "Scout: a data-parallel programming language for graphics processors," *Parallel Computing*, vol. 33, no. 10-11, pp. 648–662, 2007.

[7] Q. Chen and M. Guo, *Task Scheduling for Multi-core and Parallel Architectures: Challenges, Solutions and Perspectives*, Springer, 2017.

[8] V. Privman, *Finite size scaling and numerical simulation of statistical systems*, World Scientific Singapore, 1990.

[9] E. Guyon, JP. Hulin, L. Petit, and CD. Mitescu, *Physical hydrodynamics*, Oxford university press, 2001.

[10] LD. Landau and EM. Lifshitz, "Fluid mechanics.," *Course of Theoretical Physics*, 1987.

[11] P. Cicuta and AM. Donald, "Microrheology: a review of the method and applications," *Soft Matter*, vol. 3, no. 12, pp. 1449–1455, 2007.

[12] AM. Puertas and T. Voigtmann, "Microrheology of colloidal systems," *Journal of Physics: Condensed Matter*, vol. 26, no. 24, pp. 243101, 2014.

[13] T. Wang, Z. Liu, Y. Chen, Y. Xu, and X. Dai, "Load balancing task scheduling based on genetic algorithm in cloud computing," in *Dependable, Autonomic and Secure Computing (DASC), 2014 IEEE 12th International Conference on*. IEEE, 2014, pp. 146–152.

[14] JC. Gehrke, K. Jansen, Kraft SJ., and J. Schikowski, "A PTAS for scheduling unrelated machines of few different types," in *International Conference on Current Trends in Theory and Practice of Informatics*. Springer, 2016, pp. 290–301.

[15] V. Sels, J. Coelho, AM. Dias, and M. Vanhoucke, "Hybrid tabu search and a truncated branch-and-bound for the unrelated parallel machine scheduling problem," *Computers & Operations Research*, vol. 53, pp. 107–117, 2015.

[16] CM. Woodside and GG. Monforton, "Fast allocation of processes in distributed and parallel systems," *IEEE Transactions on Parallel & Distributed Systems*, , no. 2, pp. 164–174, 1993.

[17] TA. Waigh, "Advances in the microrheology of complex fluids," *Reports on Progress in Physics*, vol. 79, no. 7, pp. 074601, 2016.

[18] JG. Dhont, *An introduction to dynamics of colloids*, vol. 2, Elsevier, 1996.

[19] W. Paul and DY. Yoon, "Stochastic phase space dynamics with constraints for molecular systems," *Physical Review E*, vol. 52, no. 2, pp. 2076, 1995.

[20] H. Hasimoto, "On the periodic fundamental solutions of the stokes equations and their application to viscous flow past a cubic array of spheres," *Journal of Fluid Mechanics*, vol. 5, no. 2, pp. 317–328, 1959.

[21] G. Ortega, AM. Puertas, FJ. de Las Nieves, and EM Garzón, "GPU computing to speed-up the resolution of microrheology models," in *International Conference on Algorithms and Architectures for Parallel Processing*. Springer, 2016, pp. 457–466.

[22] G. Ortega, AM. Puertas, and EM. Garzón, "Accelerating the problem of microrheology in colloidal systems on a GPU," *The Journal of Supercomputing*, vol. 73, no. 1, pp. 370–383, 2017.

Arquitecturas del procesador, multiprocesadores y chips multinúcleo

Estudio sobre la paralelización de modelos MOEAs de predicción terapéutica con Toxina Botulínica tipo A en migraña

Franklin Parrales Bravo[1] Alberto Del Barrio García[2] y José Ayala Rodrigo[3]

Resumen— **El presente trabajo aborda la disminución del coste computacional en la tarea de feature weighting para los modelos predictivos de la respuesta al tratamiento de la migraña con toxina botulínica tipo A. Nos diferenciamos de un trabajo existente dado que utilizamos algoritmos evolutivos multiobjetivos (MOEA) en lugar de Simulated Annealing (enfoque de solución única). Más específicamente, consideramos los MOEA que permiten su ejecución en paralelo. Todo ello con el objetivo de mejorar los tiempos de entrenamiento de los modelos predictivos de respuesta al tratamiento. Los resultados obtenidos muestran que se obtienen precisiones cercanas al 84 % mientras que los tiempos de entrenamiento disminuyen (menos de 3 horas) cuando se usan 4, 6 y 8 hilos. Este trabajo mejora el tiempo de entrenamiento al usar Simulated Annealing para la tarea de feature weighting.**

Palabras clave— **MOEA, paralelismo, feature weighting, frente de Pareto, migraña.**

I. Introducción

Uno de los mayores problemas que enfrentan las enfermedades crónicas es su tratamiento continuo para mitigar o eliminar sus síntomas. Las enfermedades crónicas como el Parkinson o la migraña son progresivas y esto conlleva un aumento significativo en la discapacidad personal, social y laboral [1]. La migraña crónica se define como un dolor de cabeza que se presenta durante 15 o más días al mes durante más de 3 meses y que tiene las características de un dolor de cabeza de migraña al menos 8 días por mes [2]. A nivel mundial, aproximadamente el 2 % de la población experimenta migraña crónica [3]. La toxina botulínica tipo A (BoNT-A) ha sido usado como un tratamiento extendido para la migraña crónica desde su aprobación en 2010 por la FDA de los EEUU [4], [5].

En la práctica clínica, alrededor del 20-30 % de los migrañosos crónicos no responden a BoNT-A [6]. Conocer la relación fenotipo-respuesta puede ayudar [7] en el desarrollo de nuevos tratamientos para el 20-30 % de pacientes que no responden al tratamiento. Además del coste, evitaría que los pacientes sufrieran el dolor asociado con el tratamiento.

En [8] los autores logran un 85 % de precisión al predecir la respuesta a la primera y segunda infiltración del tratamiento de la migraña con BoNT-A. Aunque este trabajo proporciona una buena base para obtener modelos predictivos de la respuesta a cada infiltración del tratamiento, tiene algunas limitaciones: por un lado, no permite una optimización multiobjetivo. Con este enfoque, sería posible saber qué características médicas son importantes para todos los modelos de predicción en lugar de conocer la relevancia de un modelo de predicción específico. Por otro lado, tiene un alto coste computacional [9] como consecuencia del uso de Simulated Annealing (SA) [10], una heurística de solución única, para la tarea de feature weighting. Por lo tanto, en este artículo, pretendemos realizar un estudio del paralelismo de varios algoritmos que permiten la optimización multiobjetivo para disminuir el coste computacional al tiempo que permite la optimización multiobjetivo al predecir la respuesta al tratamiento con BoNT-A. Para ello, presentamos una comparación en términos de ejecución de tiempo, número de hilos y porcentaje de precisión de cada algoritmo multiobjetivo evaluado. En este estudio se obtuvieron porcentajes de precisión cercanos al 85 % para cada infiltración del tratamiento al aplicar los algoritmos MOEA, que es comparable al porcentaje de precisión obtenido al aplicar SA. Sin embargo, la paralelización de los MOEA reduce el tiempo de ejecución de la SA en la tarea de feature weighting, de más de 8 horas a menos de 2 cuando se emplean 8 hilos.

El resto del artículo está organizado de la siguiente manera. La sección II describe el trabajo relacionado con algunas técnicas aplicadas a la migraña y otras enfermedades. En la Sección III, se explica nuestra metodología para predecir los resultados del tratamiento. La sección IV describe los experimentos y comparaciones entre diferentes algoritmos y nuestra solución. Finalmente, nuestras conclusiones y líneas de trabajo futuras se presentan en la Sección V.

II. Trabajo relacionado

Para predecir la respuesta a cada etapa del tratamiento con BoNT-A, podemos utilizar los algoritmos de extracción de datos más comunes utilizados por la comunidad de investigación en muchos campos. Por ejemplo, de acuerdo con [11]: C4.5, k-means, Support Vector Machines (SVM), algoritmo de Esperanza-Maximización (EM), PageRank, AdaBoost, k-NN, Naive Bayes, y CART. En el trabajo presentado en [8], se ha realizado una comparación

[1]Dpto. de Arquitectura de Computadores y Automática, Univ. Complutense de Madrid. Carrera de Ing. en Sistemas Computacionales, Facultad Ciencias Matemáticas y Física, Universidad de Guayaquil, Guayaquil,Ecuador. e-mail: fparrale@ucm.es

[2]Dpto. de Arquitectura de Computadores y Automática, Univ. Complutense de Madrid, e-mail: abarriog@ucm.es

[3]Dpto. de Arquitectura de Computadores y Automática, Univ. Complutense de Madrid, e-mail: jayala@ucm.es

entre algunos algoritmos de clasificación. Los autores encontraron valores de alta precisión en la predicción de respuesta al tratamiento con BoNT-A (cerca del 85 % de precisión) con el uso del algoritmo de RandomTree en combinación con Simulated Annealing (SA) en el conjunto de datos clínicos. Sin embargo, algunos inconvenientes computacionales del trabajo anterior son: (1) el tiempo [9] que el ordenador requiere para entrenar el modelo cuando se aplica SA para un alto número de iteraciones; (2) SA usa un enfoque de solución única [10]. Este enfoque presenta algunas ventajas como la simplicidad y un bajo número de evaluaciones de funciones. Sin embargo, podría quedar atrapado en óptimos locales con alta probabilidad [12]. Además, la información no se comparte entre las soluciones candidatas cuando se aplica el enfoque de solución única, y algunos aspectos, como la convergencia prematura, el aislamiento de los óptimos y los sesgos en el espacio de búsqueda, deben tratarse con sumo cuidado. (3) Para administrar los tratamientos clínicos con múltiples etapas (como en el caso del tratamiento BoNT-A para la migraña crónica), es necesario entrenar un modelo de predicción de cada etapa por separado, dada la imposibilidad de abordarlos de manera unificada.

La metodología presentada en este trabajo hace uso de los algoritmos evolutivos multiobjetivos (MOEAs) [13], [14] porque permiten manejar múltiples objetivos de optimización al mismo tiempo que se mejora el tiempo de entrenamiento de los modelos de predicción. Además, este trabajo incorpora algunos MOEA que permiten ejecuciones en paralelo con el propósito de reducir el tiempo en el entrenamiento de los modelos de predicción de respuesta al tratamiento.

III. Metodología

La Figura 1 presenta el flujo de trabajo del experimento en el que se basa este documento. En primer lugar, se carga una base de datos con los registros médicos de los dos hospitales participantes. En segundo lugar, el atributo de clase es definido, considerando los datos retrospectivos disponibles. En tercer lugar, las características clínicas se categorizan para trabajar con datos homogéneos. Después, se aplican algunos métodos multiobjetivos paralelizables a la tarea de feature weighting. Finalmente, el algoritmo de clasificación RandomTree es aplicado para mejorar el tiempo utilizado al entrenar los modelos de predicción y tener un bajo coste computacional.

A. Datos clínicos

Se han recopilado datos médicos retrospectivos de varias historias clínicas de pacientes con tratamiento previo o actual de migraña crónica con BoNT-A con seguimiento en la unidad de cefalea de dos hospitales de nivel terciario.

El número de pacientes recogidos en dos hospitales fue de 173 (116 de *Hospital Clínico Universitario* en Valladolid y 57 de *Hospital Universitario de La Princesa*, en Madrid). Un total de sesenta y dos características básicas han sido categorizadas. Las características médicas recopiladas se relacionaron con los siguientes puntos: características clínicas del dolor, características demográficas de los pacientes, comorbilidades, fármacos preventivos probados y concomitantes, medidas de impacto del dolor y parámetros analíticos disponibles.

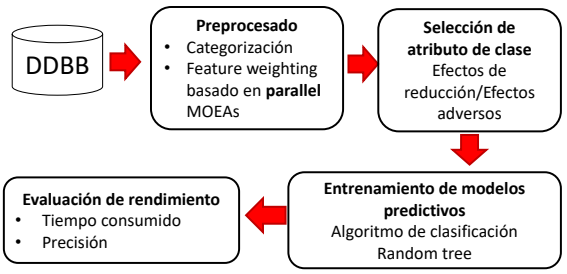


Fig. 1: Diagrama de flujo del experimento.

La Tabla I presenta un ejemplo de datos continuos y categorizados disponibles en nuestro conjunto de datos clínicos.

B. Preprocesado

B.1 Categorización de las características clínicas

Para mejorar la precisión de la predicción para el tratamiento con BoNT-A, primero se categorizan los datos heterogéneos de los hospitales. El método seleccionado para la categorización de nuestros datos médicos se basa en la media y la desviación estándar. Aplicando este método es posible trabajar con valores más homogéneos.

El tipo de categorización de desviación estándar y media centra los intervalos alrededor de la media (μ), y define intervalos subsiguientes sumando o restando la desviación estándar (σ). Por ejemplo, si tres categorías están definidas para cada una de nuestras características clínicas, los intervalos $(V_{min},\ \mu-\sigma)$, $(\mu-\sigma,\mu+\sigma)$ y $(\mu+\sigma,\ V_{max})$ son usados para referirse a valor 1, valor 2 y valor 3, respectivamente. Cabe señalar que V_{min} y V_{max} son los valores mínimo y máximo de los datos, respectivamente.

B.2 Feature weighting

Nuestro propósito es encontrar aquellas ponderaciones que mejoren la representación de las etiquetas numéricas codificadas por los médicos para cada etapa (Feature weighting). Este problema es multiobjetivo porque necesitamos encontrar los pesos óptimos que mejoren la precisión de todos los modelos predictivos de las etapas del tratamiento. La aplicación de la implementación SA de [15] para la tarea de feature weighting no resuelve simultáneamente la minimización del error de predicción para todas las etapas. Por esta razón, consideramos en este trabajo el uso de algoritmos evolutivos multiobjetivos (MOEA). El objetivo de los MOEAs es lograr un conjunto de soluciones eficientes, no dominadas o frente óptimo de Pareto [16]. Un punto importante a considerar es que los algoritmos de MOEA manejan un conjunto de soluciones (población) en lugar de una sola solución

TABLA I: Ejemplo de características continuas y categorizadas disponibles en los datos clínicos.

Edad de inicio de toxina (años)	Índice de masa corporal (kg/m^2)	Hemoglobina (g/dL)	Creatinina (mg/dL)	Plaquetas (u/mcL)	Efectos de reducción (1-4)
51	20.39	13.4	0.71	213000	4
49	26.5	14.2	0.55	252000	2
36	23.15	13.5	0.44	304000	3
26	17.7	13.1	0.66	218000	2
31	NA	14.8	0.71	327000	1
50	NA	16.2	0.74	327000	3

como en el caso de las SA. Como consecuencia de tener más soluciones, su coste computacional es mayor que los algoritmos con un enfoque de solución única, especialmente cuando se realiza sin paralelismo [17]. La paralelización permite distribuir la carga computacional en diferentes núcleos de la computadora, haciendo que la ejecución de tareas sea eficiente. Podemos usar implementaciones paralelas de MOEA para lograr una ejecución más rápida de algoritmos y un menor coste computacional [18].

$$e_i = 100 - Acc_i \ , \quad \forall i \in [1, s] \ . \tag{1}$$

Con respecto a nuestro problema, el objetivo es minimizar el error e_i para todas las etapas (s), descrito por la Ecuación 1, donde Acc_i se refiere al porcentaje de precisión del modelo de predicción correspondiente. En este documento contemplamos dos infiltraciones, por lo que hay dos objetivos que deben minimizarse simultáneamente, es decir, e_1 y e_2. El enfoque se ha implementado utilizando el framework MOEA presentado en [19]. Más específicamente, nos centraremos en el uso de MOEA que pueden ser ejecutados en paralelo para disminuir el coste computacional. Los algoritmos seleccionados son: GDE3 [20], PESA2 [21], SMPSO [22], NSGA-II [23], NSGA-III [24] and SPEA2 [25].

C. Selección de atributo de clase

Con el fin de medir la eficiencia de una infiltración del tratamiento, debemos definir el *atributo de clase*. Esto se refiere a la característica clínica utilizada para medir la efectividad del tratamiento. El valor de HIT6 [26], la intensidad, la duración y la frecuencia de los ataques [27] son buenos candidatos para ser seleccionados como atributos de clase de acuerdo a los médicos. Sin embargo, los valores de algunas de estas características no han sido proporcionados en nuestro conjunto de datos médicos. De hecho, el valor de HIT6 no se encuentra en muchos de nuestros registros clínicos. En consecuencia, hemos seleccionado como atributo de clase a la combinación de los efectos de reducción (R) y los efectos adversos (A) dado que son las características que se encuentran con mayor frecuencia en nuestra base de datos. La reducción y los efectos adversos se definen con valores proporcionados directamente por los médicos.

Estas características clínicas, R y A, son valores medibles desde un punto de vista objetivo basado en definiciones. R toma valores de 1 a 4 según el porcentaje de reducción de días de migraña. Toma el valor de 1 cuando el porcentaje de reducción de días de migraña es menor o igual a 25 %, 2 para el intervalo entre 25 % y 49 %, 3 para el intervalo entre 50 % y 74 % y 4 cuando el porcentaje es mayor o igual al 75 %. A es igual a 1 cuando no hay efectos adversos, 2 cuando hay efectos adversos leves (fácil de tolerar), 3 cuando hay efectos adversos moderados (interfieren con las actividades habituales y pueden requerir la suspensión del tratamiento) y 4 cuando hay efectos adversos graves (incapacita o deshabilita las actividades habituales y requiere la suspensión del tratamiento así como la intervención médica)

Los niveles altos de R indican buenos resultados de tratamiento, mientras que los niveles altos de A apuntan a muchos efectos adversos. Con el propósito de obtener una característica directamente proporcional, nuestro atributo de clase (N_{AC}) ha sido determinado al dividir R y A.

TABLA II: Categorización de atributos de clase

Efectos de reducción (R)	Efectos adversos (A)	R/A	Valores categorizados
1	1	1	baja
2	1	2	alta
3	2	1.5	alta
1	2	0.5	baja

En este trabajo, un enfoque similar al basado en HIT6 [26] (dos categorías de respuesta: bajo y alto) [6] ha sido considerado para la categorización de atributos de clase, en lugar de las tres categorías (bajo, Medio y alto) utilizado para el resto de las características clínicas. Las respuestas más bajas han sido etiquetadas cuando el valor N_{AC} cae en el intervalo *(V_{min}, cut-off)*, mientras que las etiquetas de respuesta alta se usan para aquellos valores que se encuentran dentro del intervalo *(cut-off, V_{max})*, donde *cut-off* se refiere al punto de corte. En este caso, $V_{min} = 0{,}25$ ocurre cuando $R = 1$ y $A = 4$, mientras que $V_{max} = 4$ ocurre cuando $R = 4$ y $A = 1$. Seleccionamos un cut-off de 1.40. La razón para usar este valor es el hecho de tratar de emular el criterio utilizado de la disminución del 30 % en el valor HIT6. El tratamiento se considera eficaz si el HIT6 se reduce en un 30 % o más, según el ensayo clínico PREEMPT [6]. De esta manera, los valores inferiores a 1.40 representan el 30 % de los valores que puede tomar N_{AC}. Luego, las categorías de respuesta al tratamiento, baja y alta, son definidas con los intervalos (0.25, 1.40) y (1.40, 4), respectivamente. La Tabla II representa una instancia del cálculo N_{AC} utilizando diferentes valores proporcionados por los hospitales.

D. Modelos de predicción de entrenamiento

Con el propósito de predecir la respuesta del tratamiento a las diferentes etapas del tratamiento con BoNT-A, varios algoritmos de clasificación han sido considerados para construir los modelos de predicción. Estos algoritmos se caracterizan por identificar categorías para los nuevos registros basados en el entrenamiento llevado a cabo con los registros anteriores (conjunto de datos de entrenamiento) [28].

En el estudio realizado por [8], han sido probados varios algoritmos de clasificación, siendo RandomTree + SA la mejor combinación con una precisión cercana al 85 % cuando se predice la respuesta a la primera y segunda infiltración del tratamiento. RandomTree es un algoritmo no determinista que construye un árbol de clasificación, considerando K características elegidas al azar para cada nodo. Los autores concluyen en su trabajo que, como resultado de ser un algoritmo no determinista, RandomTree se ha visto beneficiado con el uso de SA, dado que le ha permitido una exploración más profunda del espacio de búsqueda para evitar quedar atrapado en un mínimo local. Por ello, el algoritmo de RandomTree será considerado en nuestro experimento en combinación con los algoritmos MOEA para obtener modelos de predicción.

IV. Experimentos

En esta sección, hemos realizado una comparación de los tiempos de ejecución y las precisiones entre SA y los algoritmos de MOEA descritos en la Sección III combinados con el algoritmo de clasificación RandomTree para la tarea de feature weighting. Para completar las comparaciones, también hemos empleado el algoritmo SA utilizado por [8] para la misma tarea. Es importante tener en cuenta que la implementación de SA utilizada en ese artículo no soporta la ejecución paralela. El número de hilos considerados en los experimentos han sido: 1 (sin paralelismo), 2, 4, 6 y 8. El ordenador utilizado para realizar los experimentos contiene un procesador Intel Core i7-4790 con CPU de 3.60 GHz (4 núcleos, con 2 hilos por cada núcleo) y 16 GB de la memoria RAM. El número de iteraciones del experimento ha sido establecido en 10^6 como en [8]. El tamaño de la población para los algoritmos de MOEA ha sido establecido en 100. Este valor ha sido seleccionado para garantizar la diversidad de soluciones al mismo tiempo que se evita una convergencia lenta de las soluciones [29], [30].

A. Tiempo de ejecución

La Tabla III presenta el tiempo de ejecución de los algoritmos empleados anteriormente siguiendo el formato de hora: minutos: segundos. Dado que el método SA aplicado [15] no realiza una optimización multiobjetivo, hemos dividido el tiempo necesario para realizar la tarea de feature weighting en dos operaciones, una para cada infiltración del tratamiento BoNT-A.

De acuerdo con los resultados, podemos observar que los algoritmos paralelos de MOEA ejecutados en dos o más hilos han requerido menos tiempo que el algoritmo SA. Los MOEA se benefician del uso de más hilos para distribuir la carga computacional de la tarea de feature weighting. Sin embargo, es necesario tener en cuenta que la diferencia de tiempo entre 6 y 8 hilos es mucho menor que la diferencia entre 1, 2 y 4 hilos. Por otro lado, al parecer, podemos observar que los MOEA tienen un tiempo de ejecución más largo que SA cuando solo se ejecutan sobre 1 hilo. Sin embargo, SA solo realiza la tarea de feature weighting para una sola infiltración. Por lo tanto, el tiempo total real empleado por SA es la suma de los tiempos de la primera y segunda infiltración. Este valor es alrededor de 30 minutos más alto que el empleado por los algoritmos de MOEA con 1 hilo.

TABLA III: Tiempo de ejecución de los algoritmos de SA y MOEA en la tarea de feature weighting

Algoritmos	**Número de hilos**				
	1	**2**	**4**	**6**	**8**
SPEA2	8:16:53	4:06:08	2:30:37	2:02:08	1:54:20
NSGAIII	8:12:20	4:05:14	2:28:52	2:02:17	1:55:32
NSGAII	8:12:04	4:05:12	2:28:48	2:02:05	1:54:35
SMPSO	8:13:02	4:06:16	2:29:04	2:07:27	1:59:15
PESA2	8:16:53	4:07:01	2:29:27	2:05:33	1:58:19
GDE3	8:13:02	4:06:08	2:29:45	2:02:39	1:54:38
SA-stage 1	4:13:05	NA	NA	NA	NA
SA-stage 2	4:32:31	NA	NA	NA	NA

B. Precisión

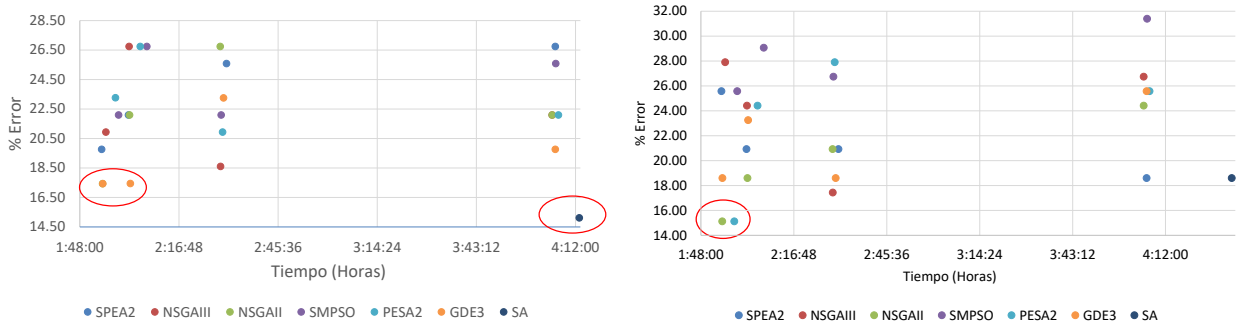
La Tabla IV presenta los mejores valores de precisión al predecir la respuesta del tratamiento a BoNT-A para la primera y la segunda infiltración. Para presentar los resultados de esta tabla, hemos seleccionado para cada algoritmo las soluciones no dominadas que tienen los errores más bajos e_1 y e_2 (primera y segunda infiltración, respectivamente) como se describe en la Sección III-B.2.

De acuerdo con los resultados, podemos observar que se obtienen valores altos de precisión, sensibilidad y especificidad cuando se aplican los métodos de MOEA y cuando se usa SA. La sensibilidad y la especificidad se refieren a la fracción de verdaderos positivos y verdaderos negativos sobre los valores predichos, respectivamente. En nuestro conjunto de datos, los valores positivos y negativos se refieren a respuestas altas y bajas, respectivamente. De acuerdo con los resultados mostrados en esta tabla, SA logra la mejor precisión (84.88 %) para la infiltración 1, mientras que GDE3 y NSGAII logran la mejor precisión (84.88 %) para la infiltración 2. En estos tres resultados, se obtuvieron 0.87 y 0.81 como valores de sensibilidad y especificidad, lo que indica un bajo número de falsos positivos y falsos negativos.

Para comparar el tiempo de ejecución y el error obtenido por cada uno de los algoritmos de MOEA contenidos en la Tabla IV, se presenta la Figura 2. Las Figuras 2a y 2b representan los errores y tiempos de ejecución producidos durante la tarea de feature weighting para la primera y segunda infiltración del tratamiento con BoNT-A, respectivamente. Además,

TABLA IV: Porcentaje de precisión de los algoritmos SA y MOEAs en combinación con el algoritmo de RandomTree.

Algoritmos	Primera infiltración			Segunda infiltración		
	Precisión	Sensitividad	Especificidad	Precisión	Sensitividad	Especificidad
SPEA2	77.91 %	0.75	0.81	79.06 %	0.83	0.75
NSGAIII	81.39 %	0.77	0.83	82.56 %	0.81	0.83
NSGAII	82.56 %	0.85	0.77	**84.88 %**	0.87	0.81
SMPSO	79.06 %	0.83	0.75	82.56 %	0.81	0.83
PESA2	76.74 %	0.82	0.71	**84.88 %**	0.87	0.81
GDE3	82.56 %	0.85	0.77	81.39 %	0.77	0.83
SA	**84.88 %**	0.87	0.81	81.39 %	0.77	0.83
Media	80.73 %	0.82	0.78	82.39 %	0.82	0.81



(a) Primera infiltración

(b) Segunda infiltración

Fig. 2: Tiempo vs Error en primera y segunda infiltración.

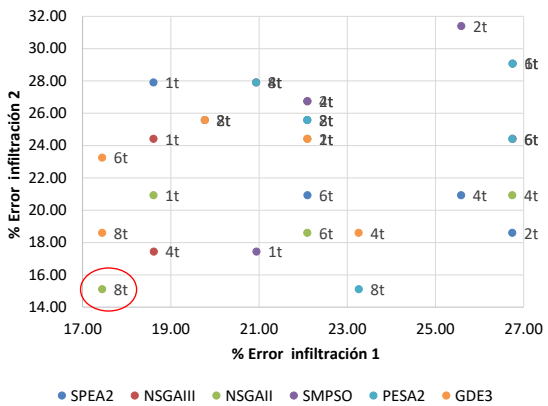


Fig. 3: Mejores puntos para cada configuración de hilos y método MOEA.

las soluciones proporcionadas por SA se presentan también en ambas figuras, ya que presentan los resultados de cada infiltración por separado. En las figuras, los mejores puntos en términos de precisión están marcados con círculos rojos. Los puntos que caen alrededor de 4, 2:30, 2 y 1:55 horas pertenecen a ejecuciones de MOEA sobre 2, 4, 6 y 8 hilos, respectivamente. Es importante observar que los gráficos muestran un mejor rendimiento para los MOEA cuando se ejecutan sobre 6 y 8 hilos que cuando lo hacen sobre 1, 2 y 4 hilos (tanto en tiempo de ejecución como en precisión), ya que el error para cada infiltración disminuye cuando cada una se considera por separado.

En la Figura 2a se puede observar que SA logra la mejor precisión (84.88 %, es decir, un error de 15.12 %) cuando solo se considera la infiltración 1. Sin embargo, La implementación de SA realizada [15] no admite la optimización multiobjetivo o el paralelismo, como se ha comentado en la Sección III-B.2. Por lo tanto, lleva más tiempo en la tarea de feature weighting (cerca de 4 horas) sin poder minimizar los errores para ambas infiltraciones al mismo tiempo, mientras que los algoritmos de MOEA pueden hacerlo. Uno de ellos, GDE3, logra un error de 17.44 % para la infiltración 1, pero obtiene un error de 18.60 % para la infiltración 2 (ver Figura 2b), siendo superado por NSGAII y PESA2 en ese escenario. Estos dos últimos obtienen las mejores minimizaciones de error para la infiltración 2 (errores de 15.2 %), pero PESA2 es superado por SA en la infiltración 1. En esta infiltración, SA no logra minimizar el error tanto como GDE3 y NSGAII, a pesar de su bajo error obtenido para la infiltración 1. Debe notarse que la solución de SA ni siquiera aparece en la Figura 2b debido a su largo tiempo de ejecución.

V. Conclusiones

Este trabajo ha estudiado las mejoras en términos de tiempo y precisión cuando se utilizan los algoritmos evolutivos multiobjetivos (MOEA) para la tarea de feature weighting. Al aprovechar el uso de múltiples subprocesos, nuestro enfoque ha necesitado menos tiempo para calcular esos pesos que una solución basada en Simulated Annealing cuando se lleva a cabo la ejecución de los MOEA. Los valores de precisión obtenidos por ambos enfoques han resultado ser bastante cercanos, por lo que podemos concluir que los MOEA han proporcionado una mejora interesante para disminuir el coste computacional al construir

modelos de predicción en el escenario de la migraña.

Por lo tanto, dada la notable reducción en el tiempo de ejecución, en el futuro podría ser beneficioso extender el uso de los MOEA para otras infiltraciones del tratamiento. Además, dados los ahorros en tiempo de ejecución, podríamos aumentar el número de iteraciones para explorar más soluciones.

Referencias

[1] Franklin Parrales Bravo, Alberto Del Barrio García, Mercedes Gallego de la Sacristana, Lydia López Manzanares, José Vivancos, and José Ayala Rodrigo, "Support system to improve reading activity in parkinson's disease and essential tremor patients," *Sensors*, vol. 17, no. 5, pp. 1006, 2017.

[2] IHS, "The international classification of headache disorders, 3rd edition (beta version)," *Cephalalgia*, vol. 33, no. 9, pp. 629–808, 2013, PMID: 23771276.

[3] JL Natoli, A Manack, B Dean, Q Butler, CC Turkel, L Stovner, and RB Lipton, "Global prevalence of chronic migraine: a systematic review," *Cephalalgia*, vol. 30, no. 5, pp. 599–609, 2010.

[4] Ninan T Mathew and Sayyed Farhan A Jaffri, "A double-blind comparison of OnabotulinumtoxinA (Botox) and Topiramate (Topamax) for the prophylactic treatment of chronic migraine: a pilot study," *Headache*, vol. 49, no. 10, pp. 1466–1478, 2009.

[5] Roger K Cady, Curtis P Schreiber, John AH Porter, Andrew M Blumenfeld, and Kathleen U Farmer, "A multicenter double-blind pilot comparison of OnabotulinumtoxinA and Topiramate for the prophylactic treatment of chronic migraine," *Headache*, vol. 51, no. 1, pp. 21–32, 2011.

[6] Stephen D Silberstein, David W Dodick, Sheena K Aurora, Hans-Christoph Diener, Ronald E DeGryse, Richard B Lipton, and Catherine C Turkel, "Per cent of patients with chronic migraine who responded per onabotulinumtoxinA treatment cycle: PREEMPT," *J Neurol Neurosurg Psychiatry*, vol. 86, no. 9, pp. 996–1001, 2015.

[7] Carlo Lovati and Luca Giani, "Action mechanisms of Onabotulinum toxin-A: hints for selection of eligible patients," *Neurological Sciences*, vol. 38, no. 1, pp. 131–140, 2017.

[8] Franklin Parrales, Alberto A Del Barrio García, Marina Gallego de la Sacristana, Mercedes, José Guerrero Peral, A, and José Luis Ayala Rodrigo, "Prediction of patient's response to OnabotulinumtoxinA treatment for migraine," *Heliyon*, vol. 5, no. 3, pp. e01043, 2019.

[9] D Janaki Ram, TH Sreenivas, and K Ganapathy Subramaniam, "Parallel simulated annealing algorithms," *Journal of parallel and distributed computing*, vol. 37, no. 2, pp. 207–212, 1996.

[10] Scott Kirkpatrick, C Daniel Gelatt, and Mario P Vecchi, "Optimization by simulated annealing," *science*, vol. 220, no. 4598, pp. 671–680, 1983.

[11] Xindong Wu, Vipin Kumar, J Ross Quinlan, Joydeep Ghosh, Qiang Yang, Hiroshi Motoda, Geoffrey J McLachlan, Angus Ng, Bing Liu, S Yu Philip, et al., "Top 10 algorithms in data mining," *Knowledge and Information Systems*, vol. 14, no. 1, pp. 1–37, 2008.

[12] Seyedali Mirjalili, Seyed Mohammad Mirjalili, and Abdolreza Hatamlou, "Multi-verse optimizer: a nature-inspired algorithm for global optimization," *Neural Computing and Applications*, vol. 27, no. 2, pp. 495–513, 2016.

[13] Carlos A Coello Coello, Gary B Lamont, David A Van Veldhuizen, et al., *Evolutionary algorithms for solving multi-objective problems*, vol. 5, Springer, 2007.

[14] David A Van Veldhuizen and Gary B Lamont, "Multiobjective evolutionary algorithms: Analyzing the state-of-the-art," *Evolutionary computation*, vol. 8, no. 2, pp. 125–147, 2000.

[15] Juan De Vicente, Juan Lanchares, and Román Hermida, "Adaptive FPGA placement by natural optimisation," in *Rapid System Prototyping, 2000. RSP 2000. Proceedings. 11th International Workshop on*. IEEE, 2000, pp. 188–193.

[16] Eckart Zitzler, Kalyanmoy Deb, and Lothar Thiele, "Comparison of multiobjective evolutionary algorithms: Empirical results," *Evolutionary computation*, vol. 8, no. 2, pp. 173–195, 2000.

[17] Juan J Durillo, Antonio J Nebro, Francisco Luna, and Enrique Alba, "A study of master-slave approaches to parallelize nsga-ii," in *2008 IEEE International Symposium on Parallel and Distributed Processing*. IEEE, 2008, pp. 1–8.

[18] Enrique Alba and Marco Tomassini, "Parallelism and evolutionary algorithms," *IEEE transactions on evolutionary computation*, vol. 6, no. 5, pp. 443–462, 2002.

[19] D. Hadka, "Moea framework," http://moeaframework.org/, 2019.

[20] Saku Kukkonen and Jouni Lampinen, "Gde3: The third evolution step of generalized differential evolution," in *Evolutionary Computation, 2005. The 2005 IEEE Congress on*. IEEE, 2005, vol. 1, pp. 443–450.

[21] David W Corne, Nick R Jerram, Joshua D Knowles, and Martin J Oates, "Pesa-ii: Region-based selection in evolutionary multiobjective optimization," in *Proceedings of the 3rd Annual Conference on Genetic and Evolutionary Computation*. Morgan Kaufmann Publishers Inc., 2001, pp. 283–290.

[22] Antonio J Nebro, Juan José Durillo, Jose Garcia-Nieto, CA Coello Coello, Francisco Luna, and Enrique Alba, "Smpso: A new pso-based metaheuristic for multi-objective optimization," in *Computational intelligence in miulti-criteria decision-making, 2009. mcdm'09. ieee symposium on*. IEEE, 2009, pp. 66–73.

[23] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and Tamt Meyarivan, "A fast and elitist multiobjective genetic algorithm: Nsga-ii," *IEEE transactions on evolutionary computation*, vol. 6, no. 2, pp. 182–197, 2002.

[24] Kalyanmoy Deb and Himanshu Jain, "An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part i: Solving problems with box constraints.," *IEEE Trans. Evolutionary Computation*, vol. 18, no. 4, pp. 577–601, 2014.

[25] Eckart Zitzler, Marco Laumanns, and Lothar Thiele, "Spea2: Improving the strength pareto evolutionary algorithm," *TIK-report*, vol. 103, 2001.

[26] Min Yang, Regina Rendas-Baum, Sepideh F Varon, and Mark Kosinski, "Validation of the headache impact test (hit-6^{TM}) across episodic and chronic migraine," *Cephalalgia*, vol. 31, no. 3, pp. 357–367, 2011.

[27] Antonio Gasbarrini, A Luca De, G Fiore, M Gambrielli, Francesco Franceschi, Veronica Ojetti, ES Torre, G Gasbarrini, Paolo Pola, and M Giacovazzo, "Beneficial effects of helicobacter pylori eradication on migraine.," *Hepato-gastroenterology*, vol. 45, no. 21, pp. 765–770, 1998.

[28] Ian H Witten, Eibe Frank, Mark A Hall, and Christopher J Pal, *Data Mining: Practical machine learning tools and techniques*, Morgan Kaufmann, 2016.

[29] Tianshi Chen, Ke Tang, Guoliang Chen, and Xin Yao, "A large population size can be unhelpful in evolutionary algorithms," *Theoretical Computer Science*, vol. 436, pp. 54–70, 2012.

[30] Eckart Zitzler and Lothar Thiele, "Multiobjective evolutionary algorithms: a comparative case study and the strength pareto approach," *IEEE transactions on Evolutionary Computation*, vol. 3, no. 4, pp. 257–271, 1999.

Diseño de un semirrestador cuántico eficiente

F. Orts[1], G. Ortega[2], E.M. Garzón[3]

Resumen— **Los computadores cuánticos basan sus operaciones en diseños de circuitos optimizados. Estos circuitos cuánticos, a diferencia de sus homólogos clásicos, están sujetos a las reglas que establece la mecánica cuántica. Actualmente, uno de los principales objetivos de la computación cuántica es conseguir implementar a gran escala el algoritmo de Shor, el cual permite factorizar números grandes en un tiempo asumible (algo impensable en computación clásica). Dicho algoritmo se basa en operaciones aritméticas, por lo tanto, optimizar dichas operaciones es fundamental. En este trabajo, se presenta un nuevo diseño de circuito semirrestador; se conoce como FGE^* y ha demostrado ser un 25 % más rápido que los semirrestadores reversibles actuales. Está basado en puertas cuánticas reversibles y no tiene salidas basura. Además, se utiliza una métrica robusta para comparar, en términos de recursos y velocidad, el circuito propuesto con los circuitos actualmente disponibles.**

Palabras clave— **Circuito restador cuántico, semirrestador cuántico, semirrestador reversible, circuitos reversibles.**

I. Introducción

La computación cuántica se centra en estudiar el procesamiento de información utilizando sistemas mecánicos cuánticos. Uno de sus objetivos es desarrollar herramientas que mejoren nuestra intuición acerca de la mecánica cuántica y hacer que esta sea más transparente e intuituva para nuestras mentes. Es bien sabido que la mecánica cuántica, que es un marco de trabajo para la construcción de teorías físicas, es contraintuitiva a pesar del hecho de que sus reglas son simples [1]. No obstante, ofrece algunas ventajas que la computación clásica no puede ofrecer [2].

La tesis de Church-Turing afirma que cualquier algoritmo se puede simular de manera eficiente utilizando una máquina de Turing [3]. Sin embargo, los algoritmos aleatorios y otros problemas no se pueden resolver de manera eficiente en una máquina de Turing determinista [4]. Inspirado por esta idea, David Deutch pudo definir una clase de máquinas que era capaz de simular eficientemente un sistema físico arbitrario. Estas máquinas son la generalización cuántica de las máquinas de Turing [5]. En base a esto, se puede afirmar que existen problemas que se pueden resolver de manera eficiente en estos computadores cuánticos, pero no en los computadores clásicos. La computación cuántica se utiliza en una amplia variedad de situaciones, como en óptica [6], coloreado de imágenes usando métodos de encriptado / desencriptado [7] o aprendizaje automático [8]. Actualmente, los mejores ejemplos en los que la computación cuántica proporciona un modelo de computación más potente que los computadores clásicos son el algoritmo de Shor para la factorización de enteros [9] y el algoritmo de búsqueda de Grover [10].

Siguiendo la idea de que los computadores cuánticos son más eficientes que los clásicos para resolver ciertos problemas, su aplicabilidad en supercomputación es de gran interés. Los computadores cuánticos se aprovechan de la mecánica cuántica para obtener características como la superposición cuántica, que permite computar eficientemente programas paralelos y distribuidos [11]. Con la superposición de n cúbits, la versión cuántica de un bit, se pueden representar y calcular 2^n posibilidades al mismo tiempo, haciendo posible nuevas formas de resolver problemas. A día de hoy no está claro qué clases de problemas se pueden resolver de manera eficiente utilizando computadores cuánticos. Sin embargo, los problemas estudiados en [9], [10] han demostrado obtener un mejor rendimiento en términos de tiempo de ejecución [1].

Los computadores cuánticos funcionan con un tipo especial de circuitos: los circuitos cuánticos. Los circuitos cuánticos se basan en operaciones básicas con puertas cuánticas, y están en consonancia con las propiedades cuánticas tales como la superposición. Incluso las operaciones complejas se basan en un conjunto de operaciones básicas [12], como sumas y restas. Por lo tanto, el uso de optimizar las operaciones básicas es importante para diseñar circuitos para así conseguir una explotación adecuada de los recursos disponibles [13]). Existen varios artículos sobre la suma y la resta (en computadores cuánticos) de dos enteros positivos [14], [15], [16], [17], [18], [19], que son las operaciones básicas más importantes. Todos estos trabajos se centran en obtener circuitos más rápidos para ser utilizados como parte de circuitos más grandes, como por ejemplo en uno que pretenda computar el algoritmo de Shor (desde el descubrimiento de Shor, muchos trabajos han investigado formas de construir circuitos cuánticos para este algoritmo [20], [21], [22], [23], [24], [25]). Por ejemplo, la resta es una operación básica para construir circuitos cuánticos para el algoritmo de Shor. Por lo tanto, desarrollar un circuito restador eficiente beneficiará a la optimización del circuito que compute el algoritmo de Shor. Un circuito semirrestador calcula la resta de dos dígitos, y hay varios trabajos que lo abordan en términos de computación cuántica [15], [18], [19], [26], [27], [28], [29].

La principal contribución de este trabajo es la descripción de un ciruito semirrestador nuevo y optimizado, que mejora la velocidad de los semirresta-

[1]Grupo de Supercomputación-Algoritmos, Dpt. de Informática, Univ. de Almería, ceiA3, 04120, Almería, España, e-mail: francisco.orts@ual.es.

[2]Dpt. de Arquitectura de Computadores, Campus de Teatinos, Univ. de Málaga, 29010, Málaga, España, e-mail: gloriaortega@uma.es.

[3]Grupo de Supercomputación-Algoritmos, Dpt. de Informática, Univ. de Almería, ceiA3, 04120, Almería, España, e-mail: gmartin@ual.es.

dores cuánticos actualmente disponibles. El circuito propuesto se basa en puertas reversibles y no tiene salidas basura. Además, se ha analizado y comparado con el resto de semirrestador utilizando una métrica justa y sólida.

El resto de este trabajo se presenta como se explica a continuación: la Sección 2 contiene una descripción de los circuitos cuánticos y las puertas cuánticas utilizadas en la construcción del semirrestador propuesto. La Sección 3 repasa los circuitos semirrestadores actualmente disponibles. La Sección 4 presenta el circuito propuesto. Finalmente, la Sección 5 resume las principales conclusiones.

II. Conceptos de circuitos cuánticos

Un circuito cuántico funciona de manera similar a uno clásico. Sin embargo, los circuitos cuánticos utilizan puertas lógicas diferentes a las clásicas, puesto que tienen que seguir varias reglas de la mecánica cuántica. Una de dichas reglas es que todas las puertas deben ser reversibles, lo que significa que a partir de la salida de una puerta (que es un estado cuántico), debe ser posible obtener la entrada (el estado cuántico anterior). Esta limitación no está presente en los computadores clásicos, que tienen un comportamiento irreversible. Es el caso de la puerta XOR, ya que no es posible obtener con exactitud la entrada a partir de la salida (varias entradas pueden dar como salida el mismo valor). La irreversibilidad no es posible en los computadores cuánticos ya que la superposición debe mantenerse a lo largo del circuito. Los circuitos cuánticos podrían tener costes adicionales a la hora de realizar ciertas operaciones clásicas debido a que necesitan transformar procedimientos no reversibles en reversibles [9].

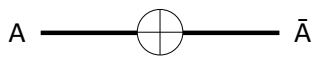
Otro tema importante es el hecho de que los circuitos cuánticos no permiten bucles. La retroalimentación de una parte de un circuito cuántico a otra no es posible, por lo que se deben encontrar nuevas formas de trabajar para resolver algunos problemas clásicos. Por otro lado, los circuitos clásicos permiten la unión de varios cables. Sin embargo, esta operación no es reversible, por lo que no se puede realizar en circuitos cuánticos por lo motivos descritos en el párrafo anterior. Además, un cúbit no se puede duplicar, por lo que no se permiten copias de un estado cuántico [1]. El valor puede ser copiado en varios casos, por ejemplo si un cúbit se establece en 1 otro se puede establecer también a 1, pero este concepto es diferente al de copiar un estado cuántico.

A. Puertas cuánticas

Las puertas cuánticas están diseñadas específicamente para ser utilizadas en computadores cuánticos. Hay puertas cuánticas que realizan las mismas operaciones que cualquier puerta clásica, pero también hay puertas cuánticas que no tienen una puerta equivalente en computación clásica. Estas puertas se pueden expresar como matrices reversibles que operan sobre un cúbit o un conjunto de cúbits, transformando sus estados iniciales en otros. Esta sección describe brevemente las puertas utilizadas en este trabajo.

A.1 Puerta Pauli-X

Hay tres puertas, llamadas puertas Pauli, que se aplican sobre un único cúbit y que son especialmente útiles y ampliamente utilizadas [30]. En este trabajo, se utiliza uno de ellas, la puerta Pauli-X, mostrada en la Fig. 1. Dicha puerta consiste en una matriz 2×2 que permuta el estado de un cúbit. La puerta Pauli-X es el equivalente en términos cuánticos de la puerta NOT clásica. Negar un estado cúbit es más complicado que negar un bit. Sin embargo, en este trabajo solo se utilizan las bases estandares como estados cuánticos, por lo que la puerta Pauli-X puede verse de manera similar a una negación clásica.

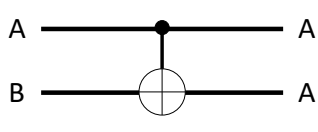


$$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

Fig. 1: Simbolo usado para representar la puerta Pauli-X y su forma matricial.

A.2 Puerta NOT-Controlada

La puerta NOT-Controlada ($CNOT$) realiza una operación sobre dos cúbits: un cúbit actúa de control y el otro es el objetivo. Su efecto es similar al de la puerta Pauli-X en el cúbit objetivo, con la diferencia de que si el cúbit de control tiene el valor 0, el cúbit objetivo no se modifica [31]. Es decir, la operación solo se realiza si el cúbit de control vale 1. En general, las puertas controladas de los circuitos cuánticos se construyen uniendo una matriz identidad en la esquina superior izquierda de la matriz de la puerta original. La representación de la puerta $CNOT$ y su forma matricial se muestran en la Fig. 2. La puerta $CNOT$ es similar a una generalización de la puerta clásica XOR [1].



$$\begin{pmatrix} 1&0&0&0 \\ 0&1&0&0 \\ 0&0&0&1 \\ 0&0&1&0 \end{pmatrix}$$

Fig. 2: Simbolo usado para representar la puerta $CNOT$ y su forma matricial.

A.3 Puertas V y V^+

Las puertas V y V^+ se presentaron en [32]. La puerta V es una puerta 1×1 que aplica la operación $V = \frac{1+i}{2}\begin{pmatrix} 1 & -i \\ -i & 1 \end{pmatrix}$ al cúbit de entrada. Por otro lado, la puerta V^+ (que también es una puerta 1×1) aplica la operación $V^+ = \frac{1-i}{2}\begin{pmatrix} 1 & i \\ i & 1 \end{pmatrix}$. La representación y forma matricial de ambas puertas se muestran en la Fig. 3.

En este trabajo solo se utilizan las bases estándares 0 y 1, por lo que hay cuatro posibilidades: $V(0), V(1), V^+(0)$ y $V^+(1)$. Los resultados de cada posibilidad se muestran a continuación:

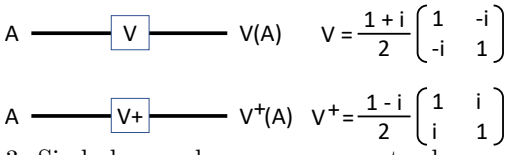


Fig. 3: Simbolos usados para representar las puertas V y V^+ y sus formas matriciales.

$$V(0) = \frac{1+i}{2}\begin{pmatrix}1 & -i\\ -i & 1\end{pmatrix}\begin{pmatrix}1\\0\end{pmatrix} = \frac{1+i}{2}\begin{pmatrix}1\\-i\end{pmatrix} \quad (1)$$

$$V(1) = \frac{1+i}{2}\begin{pmatrix}1 & -i\\ -i & 1\end{pmatrix}\begin{pmatrix}0\\1\end{pmatrix} = \frac{1+i}{2}\begin{pmatrix}-i\\1\end{pmatrix} \quad (2)$$

$$V^+(0) = \frac{1-i}{2}\begin{pmatrix}1 & i\\ i & 1\end{pmatrix}\begin{pmatrix}1\\0\end{pmatrix} = \frac{1-i}{2}\begin{pmatrix}1\\i\end{pmatrix} \quad (3)$$

$$V^+(1) = \frac{1-i}{2}\begin{pmatrix}1 & i\\ i & 1\end{pmatrix}\begin{pmatrix}0\\1\end{pmatrix} = \frac{1-i}{2}\begin{pmatrix}i\\1\end{pmatrix} \quad (4)$$

Teniendo en cuenta estas posibilidades, hay tres propiedades muy importantes que pueden derivarse de las puertas V y V^+: (1) $V(A)\times V(A) = \overline{A}$; (2) $V^+(A)\times V^+(A) = \overline{A}$; and (3) $V(A)\times V^+(A) = V^+(A)\times V(A) = A$. Estas propiedades son ampliamente usadas en diversos trabajos para simplificar y reducir circuitos cuánticos [18], [19], [33].

A.4 Puertas V-Controlada y V^+-Controlada

Estas puertas son similares a las anteriores V y V^+, pero añadiendo un cúbit de control de la misma forma que se ha explicado en la puerta $CNOT$ [32]. Por lo tanto, estas puertas son puertas 2×2 (2 entradas y 2 salidas). Mantienen las propiedades de las puertas V y V^+, pero pueden activarse o desactivarse utilizando el cúbit de control de manera conveniente. Su forma y matriz se muestra en la Fig. 4.

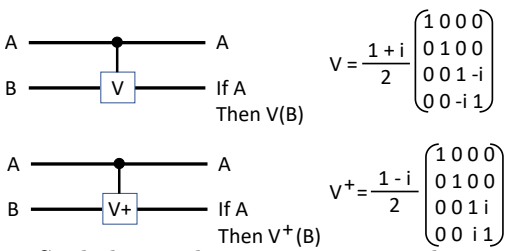


Fig. 4: Simbolos usados para representar las puertas V-Controlada y V^+-Controlada y sus formas matriciales.

A.5 Puerta Toffoli

La puerta Toffoli [34] es similar a la puerta $CNOT$, pero con dos cúbits de control en lugar de uno. Dicho de otra manera, la puerta solo opera en el cúbit objetivo si los otros 2 cúbits valen 1. Si solo se consideran las bases estándares (ortonormales) $|0\rangle$ y $|1\rangle$, la puerta Toffoli se puede usar para simular la puerta clásica $NAND$ (Fig. 6a) y también para hacer FANOUT (Fig. 6b) [1]. Teniendo en cuenta que el semirrestador propuesto en este trabajo opera con enteros, tal consideración es siempre cierta. Sin embargo, en cualquier otra situación en la que $|\alpha\rangle = \beta|0\rangle + \gamma|1\rangle$ con $\beta \neq 0$ y $\gamma \neq 0$ eso no es cierto debido a que negar un estado cuántico no es tan simple como negar un bit. La tabla de verdad para las bases estándares se muestra en la Tabla I. Actualmente, el diseño presentado en [18] es el más optimizado: usa 2 puertas V-Controladas, 1 V^+-Controlada y 2 $CNOT$. Su símbolo y matriz se muestran en la Fig. 5.

TABLA I: Tabla de verdad de la puerta Toffoli. i_1/o_1 y i_2/o_2 son los cúbits de control, y i_3/o_3 es el cúbit objetivo.

Entradas			Salidas		
i_1	i_2	i_3	o_1	o_2	o_3
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	1	0
0	1	1	0	1	1
1	0	0	1	0	0
1	0	1	1	0	1
1	1	0	1	1	1
1	1	1	1	1	0

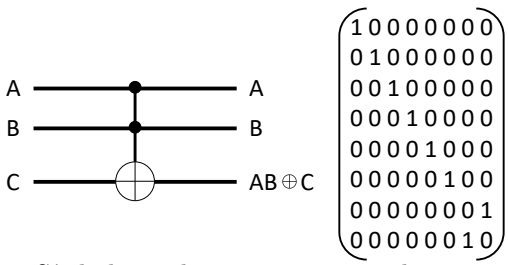
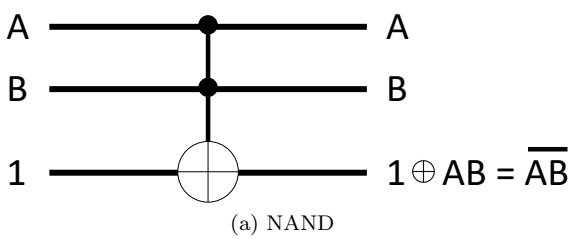
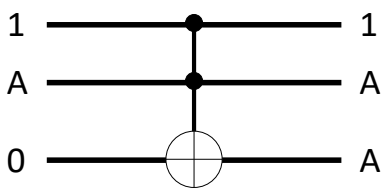


Fig. 5: Símbolo usado para representar la puerta Toffoli y su forma matricial.

Por otro lado, es importante mencionar que implementar una puerta Toffoli de control múltiple (MCT) (es decir, una puerta de Toffoli con más de 2 de cúbits de control) no es trivial. En [35] se presentó una puerta MCT que era más pequeña que las construidas anteriormente, y que no ha sido superada a



(a) NAND



(b) FANOUT

Fig. 6: (a) Una puerta $NAND$ usando una puerta Toffoli. (b) FANOUT usando una puerta Toffoli.

día de hoy. En este trabajo no se utilizan puertas MCT, por lo que no es necesario entrar en detalles.

B. Mediciones en circuitos cuánticos

Es necesario tener en cuenta varios factores importantes para analizar un circuito cuántico. Uno de ellos es el número de cúbits involucrados. Uno de los problemas más importantes a los que deben hacer frente los computadores cuánticos es la falta de recursos. Los cúbits son escasos, por lo que es importante reducir la cantidad de cúbits involucrados. En primer lugar, las entradas de un problema y también los cúbits auxiliares deben reducirse para hacer posible su computación en un simulador/ computador cuántico actual. En segundo lugar, el circuito no debe tener ninguna salida basura (es decir, cúbits que no pueden usarse al final del circuito porque es imposible conocer su valor), por lo que es necesario restaurar las salidas que no son útiles. A menos que estas salidas basura se reviertan, dichas salidas (cúbits) no se pueden usar más adelante, lo que resultaría en un desperdicio de recursos. O lo que es más grave: si están entrelazadas con entradas de otros circuitos, producirán resultados inciertos [1].

En términos de eficiencia, el parámetro más importante es el tiempo de propagación (retraso) [18]. El retraso representa la velocidad de un circuito. Se pueden utilizar varias métricas. Una de las métricas más populares es considerar que cualquier puerta tiene una unidad de retraso (1△) [36]. Siguiendo esta métrica, una puerta Toffoli (que involucra cinco puertas 2×2) tiene la misma velocidad que una puerta Pauli-X 1×1. En [37], se propone una métrica más realista, donde la velocidad de cada puerta se mide en función de su tamaño. Dicha métrica considera que el retraso de todas las puertas 1×1 y 2×2 es 1△. Siguiendo esta idea, el retraso de una puerta de $N{\times}N$ puede calcularse como su profundidad cuando se diseña con puertas de 1×1 y 2×2. Por ejemplo, la puerta Toffoli tiene un retraso de 5△, ya que está formada por 2 puertas V-Controladas, 1 V^+-Controlada y 2 $CNOT$, y ninguna de ellas se puede calcular en paralelo con otra [18]. Además, define el concepto de coste cuántico de un circuito como el número de puertas con retraso 1△ que incluye dicho circuito. El coste cuántico de la puerta Toffoli también es de 5.

Este trabajo considera la métrica presentada en [37].

III. Semirrestadores como circuitos cuánticos

Como se ha mencionado anteriormente, un semirrestador es un circuito usado para calcular la resta de dos dígitos. En el caso cuántico, realiza la resta de dos dígitos A (el minuendo) y B (el sustraendo) utilizando cúbits, que se establecen en los estados $|0\rangle$ o $|1\rangle$, por lo que sus valores pueden ser considerados como bits normales. Hay dos salidas: la diferencia $D = A \oplus B$, y el acarreo $B_{out} = \overline{\mathrm{A}}B$. La tabla de verdad se muestra en la Tabla II.

TABLA II: Tabla de verdad de un circuito semirrestador. A es el minuendo, B el sustraendo, D la diferencia y B_{out} el acarreo.

Entradas		Salidas	
A	B	D	B_{out}
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

Varios trabajos han abordado el diseño de semirrestadores en términos de computación cuántica. Sus contribuciones se centraron en reducir los recursos necesarios para construir un semirrestador y también en reducir el retraso. Como se mencionó en la sección anterior, los cúbits son un recurso limitado y, además, a mayor profundidad de un circuito, mayor es el tiempo de ejecución para calcularlo. Como también se ha mencionado ya, los circuitos cuánticos deben ser reversibles, lo que puede implicar que podrían tener costes adicionales para calcular algunos tipos de problemas. Debido a esta limitación, los circuitos deben optimizarse para evitar el desperdicio de recursos y reducir el tiempo de cálculo.

Un diseño de semirrestador se introdujo en [15]. Tiene un coste cuántico de 7 y un retraso de 7△. Necesita dos cúbits de entrada y uno auxiliar, y no produce salidas basura. El diseño del circuito consta de 2 puertas $CNOT$ y 1 Toffoli. La diferencia D se calcula utilizando solo una puerta $CNOT$, y las puertas restantes se utilizan para calcular el acarreo.

Otro semirrestador reversible se introdujo en [19], el cual mejora al anterior. Este circuito tiene un coste cuántico de 6 y un retraso de 6△. Como el anterior, usa tres cúbits como entrada (A, B y uno auxiliar), y no hay salida basura. Para conseguir esta mejora, se aprovecha de los principios descritos de las puertas V y V^+.

Se presentó un nuevo semisumador/ restador reversible en [26]. Los autores afirman que el coste cuántico de este circuito es de 4, ya que consideran cada puerta como 1△. Debido al hecho de que consta de 2 puertas $CNOT$, 1 Pauli-X y 1 Toffoli, su coste cuántico sería de 8 (el retraso también sería de 8△) siguiendo las métricas de [37]. Ya que este semirrestador también se puede usar como un sumador, se justifica este coste adicional. En [27] se presentó otro semisumador / restador, con un coste cuántico de 7, retraso de 7△ y 4 entradas. Las entradas adicionales de [26] y [27] se utilizan para elegir entre la suma y la resta. Una propuesta similar se presenta en [29] con un coste cuántico de 5, retraso de 5△ y 3 entradas.

En [18] se presentó el semirrestador más rápido disponible en la actualidad. Tiene un retraso de 4△. Además, reduce el coste cuántico a 4, manteniendo 3 cúbits de entrada y ninguna salida basura. Es una versión optimizada del circuito de [19], que utiliza el diseño de una nueva puerta llamada TR como semirrestador reversible. Dicha puerta se muestra en

la Fig. 7. Posteriormente a [18], [28] propusó un semirrestador reversible utilizando una puerta denominada DG [38] y una Pauli-X gate, como se muestra en la Fig. 12 de [28]. La puerta DG tiene un coste cuántico de 5 y un retraso de 4△. Al agregar la puerta Pauli-X, el semirrestador tiene un coste cuántico de 6 y un retraso de 5△.

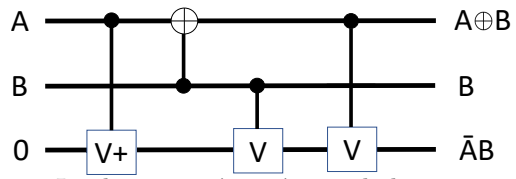


Fig. 7: Implementación cuántica de la puerta TR.

IV. Semirrestador propuesto

La puerta TR presentada en [18] tiene un retraso de 4△. Este retraso no se puede mejorar ya que las operaciones realizadas en la puerta no se pueden calcular en paralelo. La operación $A \oplus B$ y la primera puerta V-Controlada no se pueden calcular en paralelo, ya que hay dos operaciones de lectura sobre el mismo cúbit (y las reglas de la mecánica cuántica no permiten operaciones de lectura simultánea sobre un mismo cúbit). En este trabajo, hemos diseñado una nueva puerta para reordenar las operaciones de la puerta TR, posponiendo la operación $A \oplus B$. Esto será extremadamente útil para reducir el retraso. Esta nueva puerta se llama FGE, y se muestra en la Fig. 8.

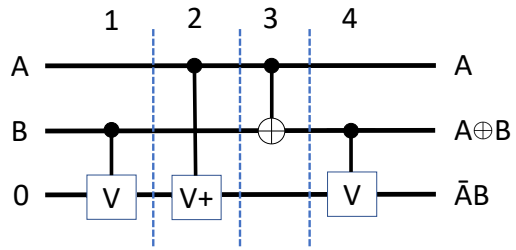


Fig. 8: Implementación cuántica de un semirrestador reversible basado en la puerta FGE. La puerta FGE tiene un retraso de 4△, el mismo que la puerta TR.

La puerta FGE tiene un coste cuántico y retraso similares a la puerta TR. Sin embargo, permite mejorar el retraso si se agrega un nuevo cúbit auxiliar Q. Los pasos 2 y 3 de la Fig. 8 no se pueden calcular en paralelo por la misma razón que se mencionó en el párrafo anterior para la puerta TR. Sin embargo, si el valor de A se transfiere al nuevo cúbit Q en el primer paso, la operación $A \oplus B$ se puede computar mediante $Q \oplus B$, que se puede calcular en paralelo con la puerta V^+-Controlada del paso 2. Así pues, el circuito obtenido, llamado FGE^*, tiene un retraso de 3△. Es necesario revertir Q para evitar una salida basura, pero esto se puede hacer en el último paso en paralelo con la última puerta V-Controlada. El circuito FGE^* se muestra en la Fig. 9.

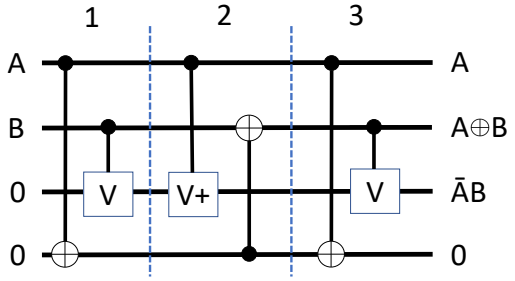


Fig. 9: Nuestra propuesta FGE^*, consistente en una optimización de la puerta FGE. Tiene un retraso de 3△. Requiere de un cúbit auxiliar extra para mejorar el retraso en 1△.

A. Evaluación de los circuitos semirrestadores

En la Tabla III se muestra una comparación detallada entre los semirrestadores estudiados en términos de retraso, coste cuántico, entradas auxiliares y salidas basura. Esta tabla muestra que la propuesta presentada en [18] es la más competitiva entre los circuitos disponibles actualmente. Centrándonos en [18], nuestra propuesta tiene dos desventajas. En primer lugar, el número de puertas. El circuito propuesto necesita dos puertas adicionales: una puerta para copiar el valor de A al nuevo cúbit auxiliar, y otra para revertir dicho cúbit. Por otro lado, necesita dos cúbits auxiliares, mientras que el circuito de [18] solo necesita uno. Sin embargo, nuestro circuito muestra mejores resultados en términos de retraso con respecto a los otros diseños de la tabla.

El porcentaje de mejora de la puerta FGE^* respecto a los diseños restantes en términos de retraso, coste cuántico, entradas auxiliares y salidas basura se muestra en la Tabla IV. La tabla muestra que FGE^* implica el uso de más puertas cuánticas que [18], [28], [29]. Además, FGE^* necesita un cúbit de entrada adicional con respecto a [15], [18], [19], [26], [28], [29] para mejorar el retraso. De manera similar a los otros circuitos, su número de salidas basura es 0. Si enfocamos nuestra atención en la columna del retraso de la Tabla IV, se puede observar que FGE^* supera incluso al más rápido ([18]). Esto significa que nuestra propuesta puede verse como el semirrestador más rápido actualmente disponible, y es la mejor opción cuando la velocidad es el factor más importante a considerar.

V. Conclusiones

En este trabajo se ha presentado un circuito semirrestador cuántico, llamado FGE^*. Es al menos un 25 % más rápido que los otros circuitos semirrestadores actualmente disponibles. Su retraso es de solo 3△, y no tiene salidas basura. El diseño del circuito ha sido explicado y estudiado en detalle. Además, se ha seguido una métrica robusta para evaluar el circuito propuesto y compararlo con los circuitos actualmente disponibles.

TABLA III: Comparativa de los diferentes diseños de semirrestradores en términos de retraso, coste cuántico, entradas auxiliares y salidas basura.

Diseño propuesto en	**Retraso**	**Coste Cuántico**	**Entradas Auxiliares**	**Salidas Basura**
(Montaser et al., [26])	8	8	1	0
(Theresal et al., [27])	7	7	2	0
(Murali et al., [15])	7	7	1	0
(Thapliyal et al., [19])	6	6	1	0
(Das et al., [28])	6	5	1	0
(Sarma et al., [29])	5	5	1	0
(Thapliyal, [18])	4	4	1	0
FGE	4	4	1	0
FGE^*	3	6	2	0

TABLA IV: Ratio de mejora de la puerta FGE^* (en %) respecto al resto de circuitos, en términos de retraso, coste cuántico, entradas auxiliares y salidas basura.

	Mejora de FGE^* (en %)			
Diseño propuesto en	**Retraso**	**Coste Cuántico**	**Entradas Auxiliares**	**Salidas Basura**
(Montaser et al., [26])	62	25	−50	0
(Theresal et al., [27])	57	14	0	0
(Murali et al., [15])	57	14	−50	0
(Thapliyal et al., [19])	50	0	−50	0
(Das et al., [28])	50	−14	−50	0
(Sarma et al., [29])	38	−14	−50	0
(Thapliyal, [18])	25	−33	−50	0

Agradecimientos

Este trabajo ha sido parcialmente financiado por el Ministerio de Ciencia de España a través del Proyecto RTI2018-095993-B-I00, por la Junta de Andalucía a través del Proyecto P12-TIC301 y por el Fondo Europeo de Desarrollo Regional (FEDER). F. Orts es becario FPI (Proyecto TIN2015-66680-C2-1-R) del Ministerio de Educación de España. G. Ortega es becaria del programa nacional "Juan de la Cierva-Incorporación"

Referencias

[1] Nielsen, M. A., Chuang, I. L. (2017). Quantum Computation and Quantum Information 10th edition. Cambridge University Press.

[2] Heilmann, R., Gräfe, M., Nolte, S., Szameit A. (2015). A novel integrated quantum circuit for high-order W-state generation and its highly precise characterization. Science bulletin, 60(1), 96–100.

[3] Goldin, D., Wegner, P. (2005). The Church-Turing thesis: Breaking the myth. In: Conference on Computability in Europe, Springer, 152–168.

[4] Papadimitriou, C. H. (2003). Computational complexity. John Wiley and Sons Ltd.

[5] Deutsch, D. (1985). Quantum theory, the Church–Turing principle and the universal quantum computer. Proc. R. Soc. Lond. A, 400(1818), 97–117.

[6] Nielsen, M. A. (2004) Optical quantum computation using cluster states. Physical review letters, 93(4), 040503.

[7] Yang, Y. G., Jia, X., Sun, S. J., Pan, Q. X. (2014). Quantum cryptographic algorithm for color images using quantum fourier transform and double random-phase encoding. Information Sciences, 277, 445–457.

[8] Lloyd, S., Mohseni, M., Rebentrost, P. (2017). Quantum algorithms for supervised and unsupervised machine learning. arXiv preprint arXiv:1307.0411.

[9] Shor, P. W. (1999). Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM review, 41(2), 303–332.

[10] Grover, L. K. (1996). A fast quantum mechanical algorithm for database search. In: Proceedings of the twenty-eighth annual ACM symposium on Theory of computing, ACM, 212–219.

[11] Back, R., Sere, K. (1992). Superposition refinement of parallel algorithms. In: Formal Description Techniques, IV, Elsevier, 475–493.

[12] Möttönen, M., Vartiainen, J. J., Bergholm, V., Salomaa, M. M. (2004). Quantum circuits for general multiqubit gates. Physical Review Letters, 93(13), 130502.

[13] Orts, F., Ortega, G., Garzón, E. (2018). A quantum circuit for solving divisions using Grover's search algorithm. In: Proc. 18th Int. Conf. Comput. Math. Method. Sci. Eng, 1–6.

[14] Gidney, C. (2018). Halving the cost of quantum addition. Quantum, (2), 74.

[15] Murali, K., Sinha, N., Mahesh, T., Levitt, M. H., Ramanathan, K., Kumar, A. (2002). Quantum-information processing by nuclear magnetic resonance: Experimental implementation of half-adder and subtractor operations using an oriented spin-7/2 system. Physical Review A, 66(2), 022313.

[16] Takahashi, Y., Kunihiro, N. (2008). A fast quantum circuit for addition with few qubits. Quantum Information & Computation, 8(6), 636–649.

[17] Takahashi, Y., Tani, S., Kunihiro, N. (2017). Quantum addition circuits and unbounded fan-out. arXiv preprint arXiv:0910.2530.

[18] Thapliyal, H. (2016). Mapping of subtractor and adder-subtractor circuits on reversible quantum gates. In: Transactions on Computational Science XXVII, Springer, 10–34.

[19] Thapliyal, H., Ranganathan, N. (2009). Design of efficient reversible binary subtractors based on a new reversible gate. In: Design of Efficient Reversible Binary Subtractors Based on a New Reversible Gate, IEEE, 1–6.

[20] Beauregard, S. (2002). Circuit for Shor's algorithm using $2N + 3$ qubits. arXiv preprint quant-ph/0205095.

[21] Fowler, A. G., Devitt, S. J., Hollenberg, L. C. (2004). Implementation of Shor's algorithm on a linear nearest neighbour qubit array. arXiv preprint quant-ph/0402196.

[22] Proos, J., Zalka, C. (2003). Shor's discrete logarithm

quantum algorithm for elliptic curves. arXiv preprint quant-ph/0301141.
[23] Takahashi, Y., Kunihiro, N. (2006). A quantum circuit for Shor's factoring algorithm using $2N+2$ qubits. Quantum Information & Computation, 6(2), 184–192.
[24] Vedral, V., Barenco, A., Ekert, A. (1996). Quantum networks for elementary arithmetic operations. Physical Review A, 54(1), 147.
[25] Zalka, C. (1998). Fast versions of Shor's quantum factoring algorithm. arXiv preprint quant-ph/9806084.
[26] Montaser, R., Younes, A., Abdel-Aty, M. (2017). New design of reversible full adder/subtractor using R gate. arXiv preprint arXiv:1708.00306.
[27] Theresal, T., Sathish, K., Aswinkumar, R. (2015). A new design of optical reversible adder and subtractor using MZI. Published at International Journal of Scientific and Research Publications (IJSRP), 5(4).
[28] Das, J. C., De, D. (2017). Reversible binary subtractor design using quantum dot-cellular automata. Frontiers of Information Technology & Electronic Engineering, 18(9), 1416–1429.
[29] Sarma, R., Jain, R. (2018). Quantum gate implementation of a novel reversible half adder and subtractor circuit. In: 2018 International Conference on Intelligent Circuits and Systems (ICICS), IEEE, 72–76.
[30] Williams C. P. (2010). Explorations in quantum computing. Springer Science & Business Media.
[31] Deutsch, D., Hayden, P. (2000). Information flow in entangled quantum systems. In: Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences, The Royal Society, (456), 1759–1774.
[32] Hung, W. N., Song, X., Yang, G., Yang, J., Perkowski, M. (2006). Optimal synthesis of multiple output boolean functions using a set of quantum gates by symbolic reachability analysis. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 25 (9), 1652–1663.
[33] Maslov, D., Dueck, G. W. (2003). Improved quantum cost for N-bit toffoli gates. Electronics Letters, 39(25), 1790–1791.
[34] Toffoli, T. (1980). Reversible computing. In: International Colloquium on Automata, Languages, and Programming, Springer, 632–644.
[35] Miller, D. M., Wille, R., Sasanian, Z. (2011). Elementary quantum gate realizations for multiple-control Toffoli gates. In: Multiple-Valued Logic (ISMVL), 2011 41st IEEE International Symposium on, IEEE, 288–293.
[36] Biswas, A. K., Hasan, M. M., Chowdhury, A. R., Babu, H. M. H. (2008). Efficient approaches for designing reversible binary coded decimal adders. Microelectronics journal, 39(12) , 1693–1703.
[37] Mohammadi, M., Eshghi, M. (2009). On figures of merit in reversible and quantum logic designs. Quantum Information Processing, 8(4), 297–318.
[38] Dehghan, B., Roozbeh, A., Zare, J. (2014). Design of low power comparator using DG gate. Circuits and Systems, 5(01), 7.

Análisis Energía-Tiempo de Redes Neuronales Convolucionales Distribuidas en Clusters Heterogéneos para clasificación de EEGs

Juan José Escobar [1], Julio Ortega [1], Miguel Damas [1], Rukiye Savran Kızıltepe [2] y John Q. Gan [2]

Resumen— **El entrenamiento de redes neuronales profundas normalmente conlleva un alto coste computacional. A día de hoy, la forma más común de procesarlas es a través del uso de GPUs por su eficiencia en los algoritmos que implementan este tipo de tareas. Sin embargo, entrenar multitud de redes neuronales, cada una con diferentes hiperparámetros es aún una tarea muy pesada. En general, los clusters cuentan con una o más GPUs por nodo que podrían ser usadas para aprendizaje profundo. Este artículo propone y analiza un procedimiento paralelo y distribuido capaz de entrenar múltiples redes neuronales convolucionales (CNNs) para clasificación de EEGs, tanto en un cluster CPU-GPU heterogéneo como en un PC de escritorio que equipa una NVIDIA TITAN Xp. El procedimiento está implementado en *C++* y usa MPI para distribuir dinámicamente los hiperparámetros a través de los nodos del cluster, los cuales son los encargados de entrenar su correspondiente CNN usando Python, Keras y TensorFlow. El algoritmo propuesto ha sido analizado considerando medidas de energía y tiempos de ejecución, mostrando que cuando se utilizan más nodos se obtiene el tiempo de ejecución más bajo y que el algoritmo escala de forma lineal. Sin embargo, el PC de escritorio obtiene los mejores resultados en términos de energía.**

Palabras clave— **Redes neuronales convolucionales · Algoritmos maestro-trabajador híbridos · Análisis energía-tiempo · Clusters CPU-GPU heterogéneos · Clasificación de EEGs**

I. Introducción

La mejora en la capacidad de cómputo de los microprocesadores ha permitido abordar cada vez problemas más complejos. No obstante, están surgiendo aplicaciones relacionadas con Big Data, la inteligencia artificial o el aprendizaje profundo que exigen a las plataformas mayor rendimiento. Así se ha extendido la práctica de utilizar las GPUs de las plataformas para la computación de propósito general (GPGPU) [1]. Las GPUs, que inicialmente fueron diseñadas para aceleración gráfica 3D, empezaron a ser usadas para específicos cálculos científicos de gran complejidad, y más tarde para un mayor rango de problemas en cooperación con las CPUs [2], [3]. Sin embargo, desde hace unos años el ratio de incremento de la potencia de cómputo está cayendo debido al paulatino agotamiento de la ley de Moore, con la I+D centrada más en la eficiencia energética por las actuales exigencias medioambientales y menos en incrementar la potencia bruta de las CPUs y GPUs. Incluso la introducción de los procesadores multi-core no ha sido suficiente para revertir esta situación ya que la mayoría de las aplicaciones no están optimizadas para usar más de un núcleo, y el uso eficiente de todos los núcleos es aún tema de investigación en las ciencias de la computación. Además, hay un problema con el crecimiento exponencial de los datos generados por las aplicaciones, el cual es imposible de abordar sin el desarrollo de nuevas técnicas de procesamiento. Por tanto, actualmente el uso de clusters capaces de distribuir el trabajo constituye el mejor enfoque para aprovechar las mejoras tecnológicas y así sobrepasar la barrera de la limitación hardware.

Existen dos métodos que son comúnmente usados para paralelizar el entrenamiento de redes neuronales en clusters: (i) paralelizar el modelo [4], donde cada nodo del sistema distribuido es responsable de computar diferentes partes de una única red (por ejemplo una capa de la red), y (ii) paralelismo de datos [5], en el cual cada nodo tiene una copia completa del modelo y se encarga de manejar una porción distinta de datos que posteriormente serán combinados. Estos dos enfoques están centrados en la paralelización de una única red neuronal. Sin embargo, el problema que se considera en este artículo es el de entrenar múltiples CNNs con diferentes hiperparámetros, y cualquiera de esos dos métodos no sería eficiente dada la gran cantidad de sincronizaciones requeridas entre los nodos. Por tanto, planteamos un enfoque diferente y más eficiente, donde cada CNN es evaluada de forma independiente en cada nodo del cluster heterogéneo, el cual involucra múltiples CPUs y otros aceleradores como GPUs. Aunque el uso de los clusters para la computación de redes neuronales ha sido objeto de estudio previos, la paralelización en una plataforma heterogénea de un procedimiento con las características de nuestra aplicación es menos frecuente en la literatura.

El artículo se organiza como sigue: La Sección II describe el problema de la clasificación de EEGs y la arquitectura de la CNN implementada para evaluar el algoritmo maestro-trabajador propuesto, detallado en la Sección III. En la Sección IV se expone el setup del trabajo experimental y se analizan sus resultados. Finalmente, la Sección V resume las conclusiones y da una idea del posible trabajo futuro.

[1]Departamento de Arquitectura y Tecnología de Computadores, CITIC, Universidad de Granada (España). E-mails: {jjescobar, jortega, mdamas}@ugr.es

[2]School of Computer Science and Electronic Engineering, University of Essex (United Kingdom). E-mails: {rs16419, jqgan}@essex.ac.uk

II. Redes neuronales convolucionales para clasificación de EEGs

En bioinformática, algunas tareas de modelado o clasificación tratan con patrones definidos por una gran cantidad de características y por tanto requieren de técnicas de selección de características que puedan eliminar los patrones con entradas redundantes, ruidosas o irrelevantes. Además, estos problemas de clasificación de alta dimensionalidad tienen que ser frecuentemente abordados con lo que se conoce como la maldición de la dimensionalidad [6], es decir, la situación que ocurre cuando el número de patrones de entrenamiento es mucho menor que el número de características en cada patrón. Una buena selección de características puede proporcionar una mejora en la precisión de los clasificadores al reducir la dimensionalidad. La clasificación de EEGs, aplicada a la imaginación de movimientos o Motor Imagery (MI), basada en tareas de interfaces cerebro-computador (BCI) [7], es una de las áreas de investigación más interesantes actualmente debido a su potencial aplicación en diferentes campos, como los juegos [8], o en la medicina donde algunos pacientes han sufridos de amputaciones y/o parálisis en alguna extremidad [9].

Las aplicaciones BCI basadas en la clasificación de EEGs, como la que se considera en este artículo, son el perfecto ejemplo de la anteriormente mencionada maldición de la dimensionalidad, la cual puede ser causada por las siguientes razones, entre otras: (i) la presencia de ruido o valores atípicos ya que las señales EEG tienen una baja relación señal-ruido; (ii) la necesidad de representar la información temporal en las características porque los patrones de señales cerebrales están relacionados con cambios en el tiempo, y (iii) el carácter no estacionario de las señales de EEG, que pueden cambiar rápidamente con el tiempo o entre experimentos. El problema es que el paradigma MI basado en BCI utiliza series de amplificaciones y atenuaciones de corta duración condicionadas por la imaginación del movimiento de las extremidades, también conocidas como desincronización relacionada a eventos (ERD) y sincronización relacionada a eventos (ERS) [10]. La tarea del análisis ERD/ERS es compleja porque las señales son débiles y ruidosas, ocurren en diferentes ubicaciones de la corteza, en diferentes instantes dentro de un ensayo y en diferentes bandas de frecuencia. Además, no hay coherencia en los patrones entre los sujetos, y éstos pueden incluso cambiar dentro de una sesión para el mismo sujeto. Este escenario suele conducir a patrones de alta dimensionalidad haciendo que la cantidad de patrones disponibles para realizar el análisis ERD/ERS sea significativamente menor que la cantidad de características.

De esta manera, es obligatorio un método de selección de características para reducir la dimensionalidad de los patrones de entrada. Sin embargo, como el tamaño del espacio de búsqueda depende exponencialmente del número de características posibles, una búsqueda exhaustiva del mejor conjunto de características es casi imposible cuando la dimensionalidad es demasiado alta y la evaluación del rendimiento se basa en la precisión de un clasificador. Incluso para un número modesto de características se han propuesto previamente procedimientos de selección de características como branch-and-bound, enfriamiento simulado, algoritmos genéticos o combinaciones de ellos [11], [12]. Actualmente, con el reciente auge en el aprendizaje profundo debido a la disponibilidad de grandes bases de datos, nuevos algoritmos de procesamiento y el crecimiento del rendimiento de las GPUs, como se explicó en la Sección I, algunas redes neuronales como las convolucionales (CNN) están comenzando a usarse para la clasificación de EEGs, la cual exige una mayor potencia de cálculo. Aunque su aplicación más común es el reconocimiento de imágenes y vídeo o el procesamiento de lenguaje natural, este tipo de red neuronal se usa también para la clasificación de EEGs debido a su buen rendimiento como ya se ha demostrado anteriormente [13]. Las CNNs pueden extraer automáticamente las características espacio-temporales más discriminatorias antes del paso de clasificación directamente de la señal original, o RAW. Por lo tanto, en este artículo se adopta una CNN bidimensional (2D-CNN) como clasificador de EEGs para evaluar el rendimiento del algoritmo paralelo propuesto. Se han considerado $N_C = 128$ diferentes combinaciones de hiperparámetros para construir diferentes arquitecturas CNN, obtenidas como resultado de combinar dos posibles valores para cada uno de los siete hiperparámetros mostrados en la Tabla I. Los parámetros son arbitrarios, aunque se han elegido para variar la complejidad de las CNNs y obtener resultados válidos y coherentes en sus salidas. La Figura 1 muestra un esquema general del procedimiento propuesto y la topología de la CNN, detallada a continuación:

TABLA I
Valores para los hiperparámetros usados en la CNN. N_F: Número de filtros en las capas convolucionales con tamaño kernel de T_KxT_K; C_O: Número de capas ocultas; C_C: Número de capas totalmente conectadas (FC) con N_N neuronas; N_E: Número de épocas

N_F	T_K	C_O	C_C	N_N	N_E	K-folds
16	3	1	1	100	50	4
32	9	2	2	200	100	6

1. La primera capa consiste en un operación 2D-convolucional cuya forma de entrada es una matriz de 240x15 ya que una señal EEG está compuesta de 240 muestras para cada uno de los 15 electrodos. La convolución aplica N_F filtros con un tamaño kernel de T_KxT_K.
2. Se aplica una operación de normalización del batch para normalizar las activaciones de la primera capa en cada batch.
3. Se aplica una operación de max-pooling de tamaño 2x1 para reducir la dimensionalidad de

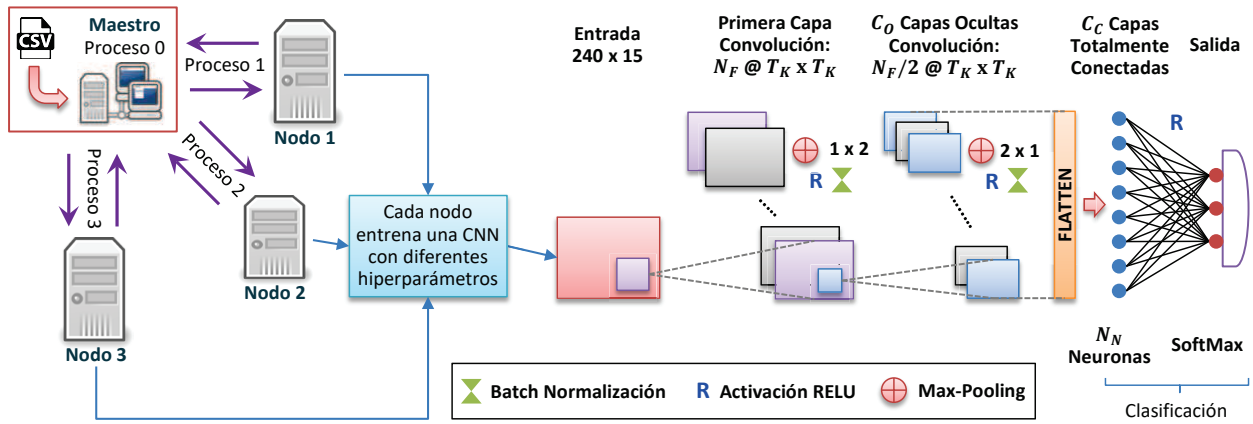


Fig. 1. Esquema del procedimiento paralelo distribuido y de la topología 2D-CNN. El maestro distribuye diferentes combinaciones de hiperparámetros a través de los nodos de cómputo. Cada uno entrena una CNN con los hiperparámetros recibidos. La CNN está compuesta por una capa 2D-convolucional seguida de C_O capas ocultas (también convolucionales), una operación de expansión o flattening, C_C capas totalmente conectadas o Fully-Connected (FC), y la capa de salida. Después de cada capa convolucional se aplica una operación de normalización del batch y otra de max-pooling. Cada capa FC contiene la mitad de neuronas de la capa anterior

las muestras para la capa convolucional previa.

4. Se emplean C_O capas ocultas, las cuales son también operaciones convolucionales. Cada convolución aplica la mitad del número de filtros usados en la capa previa pero conserva el tamaño del kernel. Después de cada capa oculta se aplica una operación de normalización del batch y otra de max-pooling de tamaño 1x2 para reducir la dimensionalidad de los electrodos. Todas las capas convolucionales componen la parte de extracción de características, la cual debería ser capaz de extraer las características espacio-temporales más relevantes de los EEGs.
5. Se usa una operación de expansión o flattening para transformar la matriz de salida de la última capa convolucional a un vector unidimensional que estará conectado a la primera capa FC.
6. C_C capas FC, que junto con la capa de salida componen la parte de clasificación. La primera capa FC tiene N_N neuronas, y sucesivamente cada capa tendrá la mitad de neuronas de la capa anterior. La última capa FC está conectada a la capa de salida y ésta sólo tiene tres unidades ya que nuestro problema de clasificación de EEGs tiene tres posibles clases.

III. Algoritmo maestro-trabajador distribuido basado en MPI

El Algoritmo 1 detalla nuestro algoritmo paralelo distribuido, esquematizado en la Figura 1. Ha sido implementado con la librería OpenMPI [14] como interfaz de paso de mensajes (MPI) para poder distribuir dinámicamente combinaciones de hiperparámetros sobre los distintos trabajadores (nodos) usados en el cluster. Empleamos este enfoque en lugar de una distribución estática para evitar el desbalanceo de carga ya que el tiempo de ejecución depende principalmente de dos factores: (i) la combinación de hiperparámetros para entrenar una red neuronal y (ii) las capacidades de cómputo de cada dispositivo ya que el cluster usado es heterogéneo. La función `Manejador` (línea 1) se divide en dos secciones: una para el proceso maestro (líneas 2-17), y otra para los trabajadores (líneas 18-27), donde cada proceso tiene un identificador único (rango) que es usado para identificarlos inequívocamente. Primero, la función recibe los parámetros de entrada necesarios para realizar el procedimiento, tales como el número total de combinaciones de hiperparámetros a evaluar (N_C), el nombre del fichero CSV con dichas combinaciones ($FichCSV$), el fichero Python con la implementación de la CNN ($FichPy$), el número de trabajadores o nodos disponibles para hacer el trabajo (N_T) y el dispositivo que ejecutará la CNN (CPU ó GPU).

El maestro (proceso MPI con rango 0) comienza a leer el fichero CSV que contiene las combinaciones de hiperparámetros (línea 3). Cada fila del fichero contiene una combinación diferente y está compuesta de múltiples columnas, que serían los hiperparámetros. En la línea 4, mediante una sentencia condicional `if-else` el programa comprueba el número total de trabajadores que están ejecutando la aplicación. Si la condición es verdadera significa que no hay disponible ningún trabajador para ayudar con la tarea y por tanto el maestro será el responsable de hacer todo el trabajo. Es decir, evaluar todas las N_C combinaciones de hiperparámetros utilizando la CPU o GPU del nodo que esté ejecutando el proceso MPI maestro. En otras palabras, el maestro está haciendo el rol del trabajador y para este caso específico es suficiente un único proceso MPI. A simple vista, esta funcionalidad parece no ser importante ya que una solución alternativa podría haber sido la de llamar al programa con un proceso MPI que actuara como maestro y otro que hiciera de trabajador. Sin embargo, esto lógicamente presenta un problema con la saturación de los recursos CPU porque cada proceso MPI se mapea a una hebra del mismo nodo y por tanto hay menos recursos CPU disponibles para entrenar la CNN. Además, hay que tener en cuenta otras sobrecargas como el paso de mensajes y la co-

Algoritmo 1: Pseudocódigo del algoritmo maestro-trabajador. El maestro dinámicamente atiende las peticiones de cada trabajador, T_j, los cuales entrenan una CNN con la combinación de hiperparámetros recibida llamando al intérprete Python. Si no se detecta ningún trabajador el maestro evaluará todas las combinaciones

```
1  Función Manejador(N_C, FichCSV, FichPy, N_T, Disp)
       Entrada: Número de combinaciones a evaluar, N_C
       Entrada: Fichero con N_C combinaciones, FichCSV
       Entrada: Fichero Python con la CNN, FichPy
       Entrada: Número de trabajadores, N_T
       Entrada: Dispositivo que ejecutará la CNN, Disp
2      si soy el Maestro entonces
           // C_i; ∀i = 1, ..., N_C:  Ej.  "<ARG_1>...<ARG_N>"
3          C ← leerCSV(FichCSV, N_C)
4          si sólo se detecta al Maestro entonces
5              repetir
6                  Cmd ← pegar("python3 ", C_i, Disp)
7                  CodigoStatus ← system(Cmd)
8              hasta que se evalúen las N_C combinaciones;
               // Distribución de combinaciones (sección MPI)
9          en otro caso
10             CargaRestante ← N_C
11             repetir
12                 C_P ← T_j ha pedido nueva combinación
13                 C_i → Se envía la combinación a T_j
14                 CargaRestante ← CargaRestante − 1
15             hasta que CargaRestante sea 0;
               // Fin de la sección MPI
16             SEÑAL_FIN → difundirSeñal(T_j)
17         fin
18     en otro caso
19         1 → T_j pide al maestro nueva combinación
20         C_l ← T_j recibe la nueva combinación
21         repetir
22             Cmd ← pegar("python3 ", C_l, Disp)
23             CodigoStatus ← system(Cmd)
24             1 → T_j pide al maestro nueva combinación
25             C_l ← T_j recibe la nueva combinación
26         hasta que el maestro envíe la SEÑAL_FIN;
27     fin
28 Fin
```

municación entre los dos procesos MPI así como sus funciones de sincronización ya que afecta al tiempo de ejecución y por tanto a la energía consumida. Todo esto ha sido considerado en nuestro algoritmo dotando al maestro la capacidad de hacer todo el trabajo sin la necesidad de más procesos.

Si la sentencia `if-else` es falsa, significa que las CNNs no serán computadas por el maestro. En este punto, maestro y trabajadores están sincronizados y listos para comenzar la sección MPI. El proceso maestro asíncronamente comienza a atender las peticiones de cada proceso trabajador y dinámicamente va distribuyendo las filas del fichero CSV hasta que no haya más trabajo por hacer (líneas 11-15). El protocolo de comunicación entre el maestro y cada trabajador para asignarle una fila es simple: una operación de recepción de la petición mediante la función MPI `Irecv` (línea 12), una operación de envío con la función MPI `Isend` (línea 13), y el decremento del contador de trabajo disponible (línea 14). Una vez que todas las N_C combinaciones (filas) han sido evaluadas el maestro envía la señal $SEÑAL\_FIN$ a los trabajadores mediante difusión (broadcast) para notificar que no hay más filas a distribuir y por lo tanto que la sección MPI ha finalizado (línea 16), permitiendo a los trabajadores acabar.

Partiendo del caso en el que se están ejecutando más de un proceso MPI, mientras que el proceso maestro está controlando la distribución de las filas/combinaciones, los trabajadores (línea 18) se encargan del entrenamiento de la CNN. Inicialmente, cada trabajador T_j solicita al maestro una combinación de hiperparámetros (línea 19) que es almacenada en C_l (línea 20). Sin embargo, también se podría haber recibido la señal $SEÑAL\_FIN$, y por tanto el bucle de la línea21 no sería ejecutado ya que eso significa que no hay trabajo disponible para este trabajador. Si este no fuera el caso, el trabajador procede de forma cíclica a realizar la tarea (línea 23), pedir al maestro más trabajo (línea 24), y esperar la respuesta (línea 25). Esto sería repetido hasta que se reciba la señal para finalizar. Por otro lado, independientemente de si el entrenamiento de la CNN lo hace el maestro o el trabajador (en caso de que no hubiese trabajadores), el procedimiento es el siguiente: desde la aplicación se realiza una llamada al intérprete Python invocando al procesador de comandos a través de la función del sistema `system` (líneas 7 y 23). Esta función necesita como argumento el comando a ejecutar, el cual está compuesto en su mayoría por los hiperparámetros y su uso es equivalente a ejecutar dicho comando directamente en una terminal de comandos del sistema operativo. Este comando es una concatenación de, por este orden: Nombre del intérprete Python, nombre del fichero Python con la implementación de la CNN, los hiperparámetros y el dispositivo que ejecutará la CNN (CPU ó GPU). Teniendo en cuenta las posibles combinaciones de siete hiperparámetros que podrían ser obtenidas de los valores mostrados en la Tabla I, un posible ejemplo del comando a ser ejecutado sería:

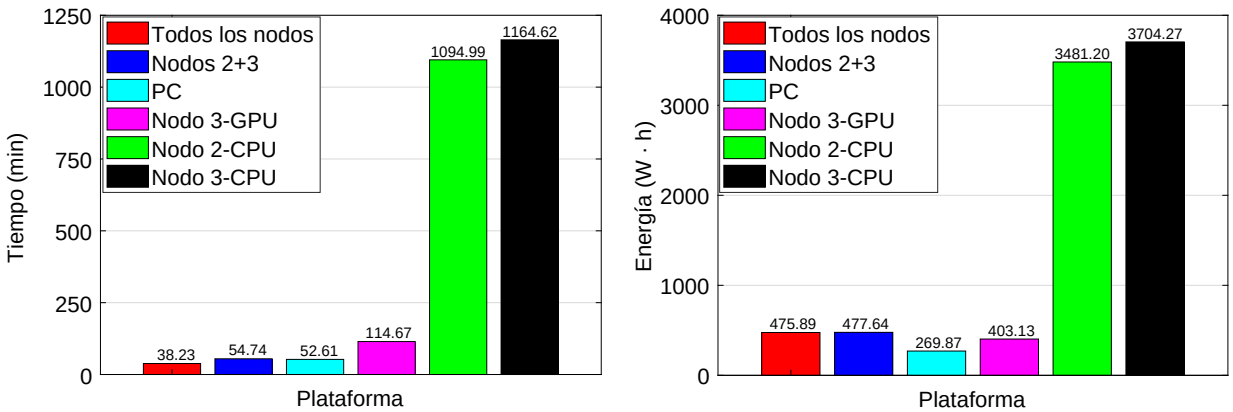
```
"python3 cnn.py 2 9 4 100 1 100 32 GPU"
```

IV. Resultados Experimentales

En esta sección se analiza el rendimiento de los códigos MPI ejecutados en un computador de escritorio (PC) con Ubuntu (18.04), y en un cluster de cuatro nodos manejado por CentOS (v7.4.1708), con 32 GB de memoria RAM y conectados por Gigabit Ethernet. El código fuente *C++* del Algoritmo 1 ha sido compilado con el compilador GNU (GCC 4.8.5), mientras que cada CNN es ejecutada por el intérprete Python (v3.6.5) y ha sido desarrollada con las APIs de Keras (v2.2.4) y TensorFlow (v1.12). Cuando se utilizan varios nodos, el nodo front-end del cluster actúa como maestro y los otros tres como trabajadores. Las características de los dispositivos CPU y GPU de cada plataforma se muestran en la Tabla II. Las medidas de energía para cada nodo se han llevado a cabo con un *watt-meter* que hemos desarrollado mediante una placa Arduino Mega. Es capaz de proveer, en tiempo real, cuatro medidas por segundo para la potencia instantánea (en Vatios) y la energía consumida acumulada (en $W \cdot h$). Las medidas de energía del *switch* que comunica los nodos están incluidas (aunque están por debajo de 5 W).

TABLA II
CARACTERÍSTICAS CPU-GPU DE LAS PLATAFORMAS USADAS EN LOS EXPERIMENTOS

Plataforma	CPU		GPU		
	Modelo	Hebras/MHz	Modelo	Núcleos/MHz	RAM/MHz
Nodo 1	2x Intel Xeon E5-2620 v2	24/2.100	Tesla K20c	2.496/706	5 GB/5.200
Nodo 2	1x Intel Xeon E5-2620 v4	16/2.100	Tesla K40m	2.880/745	12 GB/6.008
Nodo 3	2x Intel Xeon E5-2620 v4	32/2.100			
PC	1x Intel i7 4770K	8/3.500	TITAN Xp	3.840/1.582	12 GB/11.408



(a) (b)

Fig. 2. Rendimiento de las diferentes configuraciones de plataformas tras evaluar las N_C combinaciones de hiperparámetros: (a) Tiempo de ejecución; (b) Energía consumida

Para los experimentos se han utilizado 14 conjuntos de datos originales del laboratorio BCI de la Universidad de Essex, y que son descritos en [15]. Cada conjunto corresponde a un sujeto humano e incluye 178 patrones de EEG con 3.600 características, los cuales pueden pertenecer a tres tipos diferentes de movimiento o clases. Para simplificar, sólo se muestran los resultados para uno de los conjuntos de datos ya que el tiempo requerido para computar uno de ellos es el mismo. Además, no se proporciona un análisis de la calidad de las soluciones de cada CNN evaluada ya que el ámbito de este artículo está más relacionado con el comportamiento energía-tiempo del algoritmo propuesto. Sin embargo, se puede mencionar que la precisión en términos de clasificación de los hiperparámetros evaluados oscila entre el 61.81% y el 79.13%, con una media del 72.55%.

La Figura 2.a muestra el tiempo de ejecución medio que resulta de evaluar todas las posibles CNNs. Si se observan los valores correspondientes a usar combinaciones que involucra nodos del cluster (todos menos el PC), se puede ver que cuando se utilizan más nodos se obtiene el tiempo de ejecución más bajo. Ya que este es el comportamiento esperado, lo realmente interesante es la escalabilidad del programa cuando más nodos se dedican a la tarea. El tiempo de ejecución cuando se utilizan los nodos 2 y 3 debería ser, teóricamente, de la mitad del tiempo de ejecución que cuando sólo trabaja el nodo 3, pero lo que ocurre es que el tiempo es incluso menor, lo cual es conocido como aceleración super-lineal. Teniendo en cuenta que TensorFlow también ejecuta algunas operaciones en la CPU, la diferencia por tanto se debe al hecho de que la CPU del nodo 2 es más eficiente que la del nodo 3 a pesar de tener menos núcleos/hebras. Esto puede comprobarse si se observa en la figura los tiempos de ejecución en CPU de ambos nodos. El comportamiento puede explicarse si se tiene en cuenta que el número de sockets de la CPU del nodo 2 es 1, mientras que el nodo 3 tiene 2. Se ha comprobado que durante la ejecución del programa las hebras constantemente están migrando entre los diferentes núcleos del procesador, y además internamente los sockets están compartiendo la información pertinente, lo cual introduce una sobrecarga considerable. Por tanto, creemos que si se hubieran usado dos nodos exactamente iguales al nodo 3 la aceleración sería prácticamente de 2. Por último, usar todos los nodos de cómputo proporciona una reducción de tiempo igual a 3. Esto es pura coincidencia ya que siguiendo la lógica anterior la aceleración debería ser mayor de 3. Sin embargo, parece que la menor potencia de cálculo de la Tesla K20c del nodo 1 junto con la sobrecarga originada por los dos sockets CPU conlleva a una reducción del rendimiento, obteniendo una aceleración de 3. En cualquier caso parece que el algoritmo escala de forma correcta.

Por otro lado, se han comparado las ejecuciones en GPU con las equivalentes en CPU, y el resultado confirma que actualmente las CPUs no son los

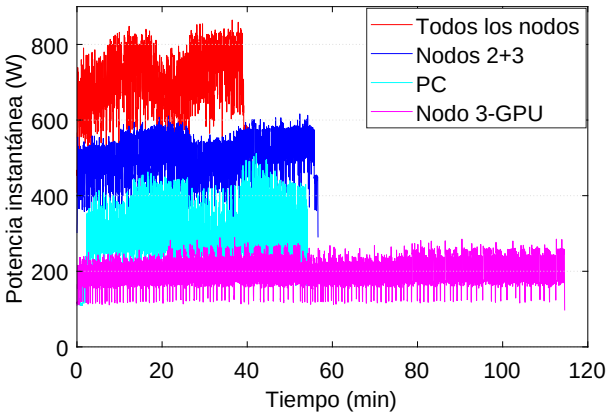


Fig. 3. Potencia instantánea de las diferentes configuraciones de plataformas tras evaluar las N_C combinaciones de hiperparámetros

mejores dispositivos para este tipo de problemas ya que el tiempo necesario para realizar la tarea es unas 10 veces más lento que el peor escenario posible al usar las GPUs. A pesar de que la CPU del nodo 3 tiene 16/32 núcleos/hebras se ha podido observar que de media el porcentaje de uso del procesador durante la ejecución no supera el 35%, por lo que la CPU no está siendo exprimida (45% para el nodo 2). Esto podría deberse a uno o varios factores: (i) que las redes neuronales no son altamente paralelizables en CPU; (ii) TensorFlow no está optimizado para estos dispositivos, que es lógico si se piensa en lo comentado en el punto (i); (iii) el modelo de la red neuronal empleado en este artículo no es lo suficientemente complejo para sacar provecho de toda la arquitectura. También se ha comprobado que el porcentaje de uso de GPU cuando se ejecuta el programa está comprendido entre el 68% y el 85%. Si se observa ahora la Figura 3, la cual indica la potencia instantánea de las distintas configuraciones de plataforma mostradas en la Figura 2 (excepto para CPU ya que tiene valores muy grandes), se puede ver que en todas las configuraciones el patrón se repite (sube y baja dos veces), y en general la tendencia es la de aumentar conforme avanza el tiempo. Esto se debe a que los hiperparámetros que construyen un modelo de red neuronal más complejo son evaluados al final de cada tramo. Si la potencia instantánea es mayor cuando el modelo es más complejo, y dado que el porcentaje de uso del dispositivo está asociado con el consumo de energía se puede concluir que al menos el punto (iii) está afectando al rendimiento, y no sólo a las CPUs sino también a las GPUs. Con respecto al rendimiento entre CPU y GPU, la CPU se ve negativamente afectada por su bajo uso y también por su menor capacidad de paralelismo de datos que el proporcionado por las arquitecturas GPU. En resumen, todo esto hace que las GPUs superen en gran medida a las CPUs y por lo tanto que la aceleración de las redes neuronales sea mucho más eficiente.

De la Figura 3 también se puede deducir que usar más nodos no sólo conlleva una reducción del tiempo de ejecución sino también un incremento de la potencia instantánea, como es lógico. Sin embargo, hay que remarcar la potencia instantánea del PC porque demuestra que esta plataforma es capaz de realizar la tarea en un tiempo ligeramente menor que el necesitado por el cluster cuando usa conjuntamente los nodos 2 y 3, pero con una potencia instantánea de prácticamente la mitad. Las razones son obvias: la arquitectura de las GPUs del cluster es más antigua (Maxwell) en comparación con la del PC de escritorio (Pascal), además de que la TITAN Xp es mucho más potente que las Teslas teniendo aproximadamente el mismo TDP (∼250 W). Además, se debe tener en cuenta que el consumo de energía de los tres nodos (nodo front-end y los nodos 2 y 3) incluye la memoria RAM, CPUs, los discos duros, placa base, etc. Esta situación demuestra que la arquitectura GPU adquiere gran importancia y, en base a los resultados de consumo de energía obtenidos podría ser mejor adquirir arquitecturas más modernas y amortizar sus costes al reducir el número de nodos y por tanto el coste monetario derivado del consumo energético.

La Figura 2.b ilustra la energía consumida (o acumulada) tras ejecutar el programa. Tal y como se esperaba, su comportamiento debería ser análogo al obtenido con los tiempos de ejecución de la Figura 2.a ya que la energía consumida depende del tiempo, de la potencia instantánea y también del número de dispositivos empleados. La forma de las barras para las ejecuciones en GPU permanecen constantes. Por tanto, lo interesante es ver cómo afecta el número de nodos y el tipo de arquitectura a la energía consumida. Lo que se aprecia es que usar todos los nodos de forma simultánea conlleva prácticamente el mismo consumo de energía que usar sólo los dos nodos más potentes (2 y 3), pero con la diferencia de que el tiempo de ejecución es menor. Usar sólo el nodo 3 causa una reducción en el consumo de energía de aproximadamente el 15.6% con respecto a las otras dos configuraciones con el cluster, pero su tiempo de ejecución es 3x más lento que al usar todos los nodos, y aproximadamente 2x si se usan los nodos 2 y 3. De nuevo, la TITAN Xp ubicada en el PC de escritorio proporciona los resultados más destacables porque es la opción con el menor consumo de energía, ofrece buenos tiempos de ejecución y es la plataforma más simple. Analizando los datos, parece que hay cierto empate entre usar el PC de escritorio o usar todos los nodos del cluster. Para clarificar esta situación se establece el producto energía-tiempo como medida de rendimiento, $P = time \cdot Energy$, y se sustituyen los datos en la Ecuación (1):

$$\begin{aligned} P_{Cluster} &= 38,23 \cdot 475,89 = 1,819327 \cdot 10^4 \\ P_{PC} &= 52,61 \cdot 269,87 = 1,419786 \cdot 10^4 \end{aligned} \quad (1)$$

Como P es menor para el PC se puede concluir que esta opción es la que da el mejor resultado, pero dependiendo del contexto quizás pueda preferirse la opción que consume menos energía o aquella que proporciona la ejecución más rápida.

V. Conclusiones

En este artículo se ha desarrollado un procedimiento paralelo maestro-trabajador distribuido que usa la librería MPI para reducir el tiempo de ejecución necesario para entrenar múltiples redes neuronales convolucionales en clasificación de EEGs. El procedimiento hace posible usar simultáneamente todos los nodos del cluster mediante comunicaciones por paso de mensajes para beneficiarse del paralelismo a nivel de dispositivos que proporcionan principalmente las GPUs de dichos nodos. También se ha implementado un esquema donde el proceso maestro es capaz de realizar todo el trabajo con el objetivo de evitar comunicaciones extra entre los procesos MPI. Se han comparado diferentes combinaciones de plataformas y alternativas paralelas analizando tanto el tiempo de ejecución como el consumo de energía. Los resultados experimentales han mostrado que cuando se utilizan todos los nodos se obtiene el mejor tiempo de ejecución y además que el programa escala de forma lineal cuando se utilizan más nodos. Por otro lado, el PC de escritorio proporciona la mejor medida para la energía consumida y el producto energía-tiempo ya que la arquitectura de la TITAN Xp que incorpora es más eficiente.

Se han evaluado 128 combinaciones de hiperparámetros para cada experimento. Aún así, normalmente el proceso de optimización en redes neuronales involucra considerar más hiperparámetros y muchas más combinaciones de ellos, aunque esto presenta el inconveniente de que el espacio de búsqueda se convertiría en un problema NP-duro. Por tanto, al observar los altos tiempos de ejecución obtenidos en los resultados experimentales de la Sección IV, está claro que es necesario diseñar estrategias de paralelización para clusters con múltiples nodos para aprovechar las GPUs que incluyen ya que en el futuro próximo la complejidad de las redes neuronales y el volumen de datos a procesos incrementará.

Respecto a la clasificación de EEGs, de cara a evaluar la escalabilidad de programa paralelo distribuido las combinaciones de hiperparámetros para las CNNs se han prefijadas de antemano. Sin embargo, la forma correcta de entrenar una red neuronal es optimizando los hiperparámetros. Por tanto, como trabajo futuro proponemos un método que combine un algoritmo de optimización capaz de seleccionar automáticamente los mejores hiperparámetros para la CNN (como un algoritmo genético), y el algoritmo distribuido propuesto en este artículo para paralelizar y explorar el máximo número de combinaciones de hiperparámetros lo más rápido posible.

Agradecimientos

Este trabajo ha sido financiado por el Ministerio de Ciencia, Innovación y Universidades y los fondos ERDF a través del proyecto PGC2018-098813-B-C31. También agradecemos al laboratorio BCI de la Universidad de Essex por el acceso a las bases de datos y a la Corporación NVIDIA por la donación de la TITAN Xp usada en esta investigación.

Referencias

[1] J.D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Krüger, A.E. Lefohn, and T.J. Purcell, "A survey of general-purpose computation on graphics hardware," *Computer Graphics Forum*, vol. 26, no. 1, pp. 80–113, 2007.

[2] P. Collet, "Why gpgpus for evolutionary computation?," in *Massively Parallel Evolutionary Computation on GPGPUs*, S. Tsutsui and P. Collet, Eds., Natural Computing Series, pp. 3–14. Springer, 2013.

[3] K. Raju and N.C. Niranjan, "A survey on techniques for cooperative cpu-gpu computing," *Sustainable Computing: Informatics and Systems*, vol. 19, pp. 72–85, 2018.

[4] J. Wawrzynek, K. Asanovic, B. Kingsbury, D. Johnson, J. Beck, and N. Morgan, "Spert-ii: a vector microprocessor system," *Computer*, vol. 29, no. 3, pp. 79–86, 1996.

[5] Y. Zou, X. Jin, Y. Li, Z. Guo, E. Wang, and B. Xiao, "Mariana: Tencent deep learning platform and its applications," *VLDB*, vol. 7, no. 13, pp. 1772–1777, 2014.

[6] R.E. Bellman, *Adaptive Control Processes: A Guided Tour*, Princeton University Press, 1961.

[7] J.S. Brumberg, J.D. Burnison, and K.M. Pitt, "Using motor imagery to control brain-computer interfaces for communication," in *Proceedings of the 10th International Conference on Augmented Cognition*, Toronto, Canada, July 2016, AC'2016, pp. 14–25, Springer.

[8] R. Wei, X.H. Zhang, X. Dang, and G.H. Li, "Classification for motion game based on eeg sensing," *ITM Web Conferences*, vol. 11, pp. 05002, 2017.

[9] N. Birbaumer and L.G. Cohen, "Brain-computer interfaces: communication and restoration of movement in paralysis," *The Journal of Physiology*, vol. 579, no. 3, pp. 621–636, 2007.

[10] G. Pfurtscheller, "Eeg event-related desynchronization (erd) and event-related synchronization (ers)," *Electroencephalography and Clinical Neurophysiology*, vol. 103, no. 1, pp. 26, 1997.

[11] J. Ortega, J. Asensio-Cubero, J.Q. Gan, and A. Ortiz, "Classification of motor imagery tasks for BCI with multiresolution analysis and multiobjective feature selection," *BioMedical Engineering OnLine*, vol. 15, no. 1, pp. 149–164, 2016.

[12] J. Ortega, A. Ortiz, P. Martín-Smith, J.Q. Gan, and J. González, "Deep belief networks and multiobjective feature selection for BCI with multiresolution analysis," in *Proceedings of the International Work-Conference on Artificial Neural Networks*, Cádiz, Spain, June 2017, IWANN'2017, pp. 28–39, Springer.

[13] Y.R. Tabar and U. Halici, "A novel deep learning approach for classification of EEG motor imagery signals," *Journal of Neural Engineering*, vol. 14, no. 1, pp. 016003, 2016.

[14] The Open MPI Project, "Openmpi documentation," `https://www.open-mpi.org/doc/`, Accessed: 2018-11-19.

[15] J. Asensio-Cubero, J.Q. Gan, and R. Palaniappan, "Multiresolution analysis over simple graphs for brain computer interfaces," *Journal of Neural Engineering*, vol. 10, no. 4, pp. 21–26, 2013.

Barreras Especulativas con Memoria Transaccional

Manuel Pedrero, Ricardo Quislant, Eladio Gutiérrez, Emilio L. Zapata, and Óscar Plata [1]

Resumen— **La Memoria Transaccional (TM) es una alternativa al modelo de programación basado en locks que pretende simplificar la programación paralela. TM sustituye locks por transacciones para resolver el problema de la exclusión mutua. Las transacciones se ejecutan de manera optimista, en paralelo, mientras el sistema transaccional comprueba si hay conflictos entre ellas. En este trabajo proponemos el uso de transacciones para implementar una barrera especulativa (SB) optimista que reemplace las barreras pesimistas basadas en locks. SBs aprovecha el soporte TM hardware para permitir que los hilos salten la barrera y ejecuten especulativamente. Cuando un hilo llega a una barrera abre una transacción para proteger la ejecución y poder volver a la barrera en caso de conflicto. Cuando el último hilo alcanza la barrera los hilos especulativos pueden acometer los cambios.**

Las contribuciones de este trabajo son: una API para las SBs implementada con extensiones TM de un sistema real (IBM Power8); un procedimiento para comprobar el estado especulativo de los hilos entre barreras para usar en códigos no transaccionales; un conjunto de directrices para el uso de las SBs. Los resultados muestran un incremento en el rendimiento de hasta 6× sobre las barreras tradicionales para algunas configuraciones de las aplicaciones evaluadas.

Palabras clave— **Barreras Especulativas, Memoria Transaccional, Memoria Compartida, IBM Power8.**

I. Introducción

La Memoria Transaccional (TM) [1] trata de simplificar la programación paralela con el uso de transacciones en lugar de locks. Con locks, una sección crítica se resuelve haciendo esperar a los hilos mientras sólo uno de ellos la ejecuta. Con TM se apuesta por una solución optimista en la que todos los hilos ejecutan la sección crítica en paralelo y sólo se serializa la ejecución si hay algún conflicto. Las transacciones que delimitan un código proporcionan atomicidad y aislamiento en la ejecución de ese código. El sistema transaccional ejecuta las transacciones especulativamente y mantiene un registro de las posiciones de memoria accedidas para detectar y resolver conflictos. En los último años se han propuesto varios sistemas TM hardware (HTM) [2], [3], [4] y los principales fabricantes de chips han añadido extensiones HTM a sus arquitecturas [5], [6], [7]. Estas extensiones se llaman *best-effort* ya que no ofrecen garantías para la finalización de las transacciones, proporcionando un mecanismo de *fallback* [8] para este caso.

En este trabajo se proponen barreras especulativas (SB) optimistas basadas en transacciones para sustituir el sistema pesimista tradicional basado en locks. Cuando un hilo llega a una SB se inicia una transacción que mantiene la ejecución aislada del sistema mientras este comprueba si hay conflictos con otros hilos. Una vez que el último hilo ha llegado a la SB, los hilos especulativos tratan de acometer los cambios realizados. Las SBs implementan un orden parcial entre las transacciones previas a la SB y las transacciones especulativas donde las últimas tienen que esperar a que finalicen las primeras para realizar el commit. Aquí se propone una implementación de las SBs con HTM. Para ello se requiere una característica indispensable del sistema HTM, que es el soporte para acciones escapadas [9]. Las acciones escapadas permiten el acceso no transaccional a memoria desde dentro de una transacción rompiendo así el aislamiento y la atomicidad de la transacción. Su utilización permite mantener la comunicación entre transacciones SB sin que se produzcan abortos.

Este trabajo hace las siguientes contribuciones sobre un trabajo preliminar [10]:

- Proporcionamos un API para las SBs implementado con extensiones HTM. El API incluye wrappers para las instrucciones de comienzo y commit de transacción, así como para el reemplazo de las barreras tradicionales por SBs. Se proporciona un procedimiento para la última barrera que termina la especulación de barreras previas y no empieza nuevas transacciones SB.
- Proponemos un procedimiento para comprobar el estado especulativo entre barreras para ser usado en códigos sin transacciones. Este procedimiento también se puede utilizar para incrementar el número de comprobaciones en códigos transaccionales.
- Proporcionamos un conjunto de directrices para el uso de SBs derivadas de nuestra experiencia usando SBs en una variedad de aplicaciones intensivas en el uso de barreras.
- Evaluamos nuestra propuesta en una máquina real IBM Power8 server.

Los resultados muestran una mejora general en el rendimiento de las SBs sobre las barreras tradicionales. La aceleración de los códigos crece con el número de hilos, especialmente debido a los dos sockets de la máquina Power8 cuando los hilos de las aplicaciones se comunican entre sockets. Las SBs con el procedimiento de comprobación de especulación mejoran el rendimiento sobre la aplicación paralela desde 2× a 6× a partir de 16 hilos y hasta 128.

II. Barreras Especulativas

A. Motivación

Las aplicaciones que hacen un uso intensivo de barreras [11] pueden mostrar un bajo rendimiento de-

[1]Departamento de Arquitectura de Computadores, Universidad de Málaga, España, 29071. E-mail: {mpedrero,quislant,eladio,zapata,oplata}@uma.es

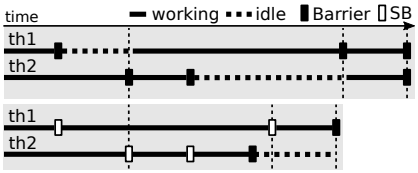


Fig. 1. Barreras tradicionales (arriba) versus SBs (abajo).

bido a (i) el desbalanceo de barrera, donde los hilos más rápidos tienen que esperar a los más lentos para continuar su ejecución; y (ii) la contención en la comunicación de barrera, especialmente en procesadores multi-chip. Las SBs están pensadas para que los hilos más rápidos especulen al llegar a la barrera en lugar de esperar. De este modo, el desbalanceo de barrera se puede utilizar para ocultar la contención de barrera. La Fig. 1 muestra un escenario donde el desbalanceo causa un retraso con las barreras tradicionales (arriba) mientras que las SBs (abajo) dan un rendimiento mejor incluso cuando el hilo más rápido tiene que esperar en la última barrera. La Fig. 1 asume que no hay conflictos de memoria entre los dos hilos. En presencia de conflictos los hilos deben sincronizarse para solucionarlos. SBs utiliza HTM para este propósito.

B. Descripción

Las SBs están ideadas para su uso con códigos transaccional donde una transacción se puede encontrar entre barreras. En esencia, las SBs hacen que la barrera se elimine y en su lugar se abra una transacción (la transacción SB). A continuación, si se encuentra una transacción en el código después de la barrera esta se ignora. La instrucción de comienzo de transacción no se ejecuta y la instrucción de commit se sustituye por una comprobación de la especulación. La especulación continuará hasta que (i) se detecta un conflicto, en cuyo caso se abortará la transacción SB, (ii) se llegue al límite de especulación, que se establece debido al número limitado de recursos del sistema HTM, y por lo tanto el hilo ha de esperar a los demás hilos, o (iii) todos los hilos hayan cruzado la SB, y la transacción SB acometerá los cambios realizados.

El API que implementa las SBs se muestra en la Fig. 2. El procedimiento SPEC_BARRIER está indicado para sustituir a las barreras tradicionales; LAST_BARRIER es similar a SPEC_BARRIER pero bloquea cualquier especulación subsiguiente. Se usa en casos para los que estamos seguros de que la especulación no tendrá éxito. TX_START y TX_STOP reemplazan a las llamadas de HTM para empezar y terminar una transacción, e incluyen código para integrarlas con las SBs. El orden parcial entre transacciones se implementa por medio de la variable *sbOrder* en la línea 2, que mantiene el orden global de la ejecución. Dicho orden se incrementa siempre que el último hilo alcanza una SB (línea 68), permitiendo que los hilos especulativos puedan acometer sus cambios. Cada hilo mantiene un orden local en la variable *order* (línea 4) que se incrementa siempre que el hilo pasa una SB. De esta manera el hilo se considera especulativo si su orden local es mayor que *sbOrder*, y puede hacer commit en caso contrario. La bandera local *spec* (línea 6) representa si el hilo está en estado especulativo.

SPEC_BARRIER: Este procedimiento implementa una SB (línea 54). Cualquier hilo no especulativo que llega a una SB incrementa su *order* local (línea 65). El último hilo incrementa *sbOrder* para generar un nuevo orden global y no inicia una transacción SB. Sin embargo, los otros hilos, que quedarían bloqueados en una barrera tradicional, continúan la ejecución abriendo una transacción SB (línea 83). Un aborto de esta transacción vuelve al hilo a la línea 70, donde el hilo comprueba si la especulación debe terminar (líneas 71 a 74), si no, el nivel de especulación se reajusta (líneas 76 to 81). especulación se limita a una SB por hilo, por lo que si se llega a otra SB el hilo espera a que su orden local sea igual a *sbOrder*. Esto se hace en una espera activa escapada para evitar abortos debido al acceso a *sbOrder* (líneas 56 to 59). Tras esto el hilo puede acometer sus cambios.

LAST_BARRIER: este procedimiento es similar a SPEC_BARRIER, pero no permite especulación tras su ejecución. Se debe usar como última barrera del hilo o cuando lo que hay después es alguna instrucción no compatible con transacción como reserva de memoria, llamadas al sistema, etc.

TX_START: este procedimiento reemplaza a la primitiva HTM que inicia una transacción (*xBegin()*), y añade soporte para SBs. Primero comprueba si el hilo ya está especulando, en cuyo caso no se hace nada puesto que ya está dentro de transacción. Si no está especulando, se sigue el procedimiento normal para abrir una transacción. Aquí se ha utilizado un fallback con eager subscription [8] que consta de un lock global que se ejecuta tras MAX_RETRIES abortos.

TX_STOP: este procedimiento reemplaza a la primitiva HTM que hace el commit de una transacción (*xCommit()*). Si el hilo no estaba especulando (líneas 22 a 30) se ejecuta el procedimiento estándar para hacer commit y se resetean las variables del descriptor. En el caso de estar especulando se comprueba si el hilo puede hacer commit (línea 32). La comprobación del orden (línea 31) debe estar escapada para que no aborte la transacción por una actualización de *sbOrder*. En el caso de que no se pueda hacer commit, la transacción SB continúa hasta que se llegue al máximo nivel de especulación, en cuyo caso queda bloqueada.

La Fig. 3 muestra un escenario con SBs y dos hilos. Las líneas punteadas representan el orden global. Primero, `th1` y `th2` ejecutan transacciones (en `tx` #, # es el orden local) antes de la primera barrera. La bandera *spec* está a 0 por lo que las transacciones empiezan y acaban con normalidad. A continuación `th1` alcanza una SB, incrementa su orden local, decrementa *sbBarrier* y comprueba que no es el último hilo en cruzar la barrera. Por lo tanto, `th1` empieza una transacción SB. Para cuando `th2` alcanza la barrera, `th1` ya ha ejecutado una transacción, que no se

```
1:  global sbBarrier ← #TH                          ▷ Barrier counter
2:  global sbOrder ← 1                              ▷ Global order for barriers
3:  thread local tx {                               ▷ Transaction descriptor
4:    order ← 1                                     ▷ Local order for the transaction
5:    retries ← 0                                   ▷ Retry count for the transaction
6:    spec ← 0                                      ▷ Flag to signal SB mode
7:    specMax ← MAX_SPEC                            ▷ Maximum spec level
8:    specLevel ← MAX_SPEC                          ▷ Current spec level
9:  }
10:
11: procedure TX_START(tx)
12:   if (tx.spec = 0) then                         ▷ Not executing a SB
13:     tx.retries ← tx.retries + 1
14:     if (tx.retries > MAX_RETRIES) then
15:       fallbackBegin()                           ▷ Acquire fallback lock
16:     else
17:       xBegin()                                  ▷ Go to line 13 on abort
18:     end if
19:   end if      ▷ If the thread is in SB mode, do nothing
20: end procedure
21: procedure TX_STOP(tx)
22:   if (tx.spec = 0) then                         ▷ Not in SB mode
23:     if (tx.retries ≤ MAX_RETRIES) then
24:       xCommit()
25:     else
26:       fallbackEnd()                             ▷ Release fallback lock
27:     end if
28:     tx.retries ← 0                              ▷ Reset tx descriptor
29:     tx.specLevel ← tx.specMax
30:   else                                          ▷ In SB mode
31:     beginEscape()
32:     if (tx.order ≤ sbOrder) then
33:       escapeEnd()
34:       xCommit()
35:       tx.retries ← 0                            ▷ Reset tx descriptor
36:       tx.specLevel ← tx.specMax
37:       tx.spec ← 0
38:     else
39:       escapeEnd()
40:       tx.specLevel ← tx.specLevel - 1;
41:       if (tx.specLevel = 0) then
42:         escapeBegin()
43:         while (tx.order > sbOrder) do
44:         end while                               ▷ Busy waiting
45:         escapeEnd()
46:         xCommit()
47:         tx.retries ← 0                          ▷ Reset tx descriptor
48:         tx.specLevel ← tx.specMax
49:         tx.spec ← 0
50:       end if
51:     end if
52:   end if
53: end procedure
54: procedure SPEC_BARRIER(tx)
55:   if (tx.spec = 1) then                         ▷ Executing a SB
56:     escapeBegin()
57:     while (tx.order > sbOrder) do
58:     end while                                   ▷ Busy waiting
59:     escapeEnd()
60:     xCommit()
61:     tx.retries ← 0                              ▷ Reset tx descriptor
62:     tx.specLevel ← tx.specMax
63:     tx.spec ← 0
64:   end if
65:   tx.order ← tx.order + 1;
66:   if ((sbBarrier ← sbBarrier - 1) = 0) then     ▷ Atomic
67:     sbBarrier ← #TH
68:     sbOrder ← sbOrder + 1                       ▷ Atomic
69:   else
70:     tx.retries ← tx.retries + 1
71:     if (tx.order ≤ sbOrder) then ▷ Do not begin a SB
72:       tx.retries ← 0                            ▷ Reset tx descriptor
73:       tx.specLevel ← tx.specMax
74:       tx.spec ← 0
75:     else
76:       if (tx.retries > MAX_RETRIES) then
77:         if tx.specMax > 1 then
78:           tx.specMax ← tx.specMax - 1
79:         end if
80:         tx.specLevel ← tx.specMax
81:       end if
82:       tx.spec ← 1
83:       xBegin()                                  ▷ Go to line 70 on abort
84:     end if
85:   end if
86: end procedure
87: procedure LAST_BARRIER(tx)
88:   if (tx.spec = 1) then                         ▷ Executing a SB
89:     escapeBegin()
90:     while (tx.order > sbOrder) do
91:     end while                                   ▷ Busy waiting
92:     escapeEnd()
93:     xCommit()
94:     tx.retries ← 0                              ▷ Reset tx descriptor
95:     tx.specLevel ← tx.specMax
96:     tx.spec ← 0
97:   end if
98:   tx.order ← tx.order + 1;
99:   if ((sbBarrier ← sbBarrier - 1) = 0) then     ▷ Atomic
100:    sbBarrier ← #TH
101:    sbOrder ← sbOrder + 1                       ▷ Atomic
102:  else
103:    while (tx.order > sbOrder) do
104:    end while                                   ▷ Busy waiting
105:  end if
106: end procedure
```

Fig. 2. Implementación de la API para las barreras especulativas (SBs)

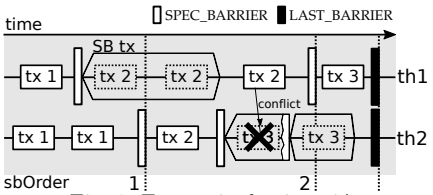


Fig. 3. Escenario de ejecución.

ha abierto realmente ya que está dentro de la transacción SB, de ahí la línea punteada. Sin embargo, th1 no puede acometer los cambios hasta el final de su segunda transacción, puesto que la comprobación del orden se realiza en el procedimiento de commit y para el commit de su primera transacción el orden local era mayor que el global. Luego, th2 ejecuta una transacción y llega a otra barrera donde abre una transacción SB. Dicha transacción es abortada debido a un conflicto con una transacción normal de th1 y es reiniciada. Se puede ver cómo cada hilo cruza las SBs sin esperar al otro hilo, excepto cuando llegan a LAST_BARRIER.

C. SBs y código no transaccional

La API que proponemos se puede usar con códigos no transaccionales. Sin embargo, las transacciones SB pueden acabar siendo muy largas para un sistema HTM best-effort si las barreras están muy separadas. Cuando hay transacciones entre las barreras el procedimiento TX_STOP comprueba el orden global y termina la transacción SB si procede. En ausencia de transacciones carecemos de este punto de comprobación intermedio. Por ello, proponemos el procedimiento CHECK_SPEC (Fig. 4).

CHECK_SPEC es básicamente el *else* de TX_STOP. La línea 2 comprueba si el hilo está ejecutando una transacción SB, e intenta terminarla si así es. Para este fin, se comprueba que el orden local es menor o igual que el global. Si no, la transacción SB continúa.

```
1:  procedure CHECK_SPEC(tx)
2:    if (tx.spec = 1) then                          ▷ In SB mode
3:      beginEscape()
4:      if (tx.order ≤ sbOrder) then
5:        endEscape()
6:        xCommit()
7:        tx.retries ← 0                  ▷ Reset tx descriptor
8:        tx.specLevel ← tx.specMax
9:        tx.spec ← 0
10:     else
11:       endEscape()
12:       tx.specLevel ← tx.specLevel - 1;
13:       if (tx.specLevel = 0) then
14:         beginEscape()
15:         while (tx.order > sbOrder) do
16:         end while                          ▷ Busy waiting
17:         endEscape()
18:         xCommit()
19:         tx.retries ← 0                  ▷ Reset tx descriptor
20:         tx.specLevel ← tx.specMax
21:         tx.spec ← 0
22:       end if
23:     end if
24:   end if
25: end procedure
```

Fig. 4. Procedimiento para comprobar la especulación.

```
1:  for (i ← 0; i < N; i ← i+1) do
2:    TX_START(tx)    ▷ Not necessary when using check_spec
3:    for (j ← 0; j ≠ L*((tid+1) %2); j ← j+1) do
4:      dummy ← dummy+1
5:    end for
6:    TX_STOP(tx) / CHECK_SPEC(tx)
7:    SPEC_BARRIER(tx)
8:  end for
9:  LAST_BARRIER(tx)
```

Fig. 5. Microbenchmark Barrier.

Nótese que se mantiene la comprobación de *specLevel* en la línea 13 aún en ausencia de transacciones entre barreras. De esta manera podemos controlar la longitud de la especulación. CHECK_SPEC se puede usar también en códigos con transacciones para aquellos casos en que el código entre transacciones sea muy largo.

D. Uso de las Barreras Especulativas

El microbenchmark Barrier de la Fig. 5 representa un escenario en el que no hay conflictos entre barreras [12]. Lo usamos para comprobar que la API propuesta no introduce conflictos adicionales relacionados con la implementación. Los hilos ejecutan cómputo local (líneas 3 a 5) y se sincronizan en una barrera (línea 7). La condición en la línea 3 implica que sólo la mitad de los hilos ejecuta el cómputo en cada iteración creando así un desbalanceo. En este escenario el uso de SPEC_BARRIER permite que los hilos ociosos continúen reduciendo el desbalanceo de barrera.

Cholesky y Recurrence se corresponden con el segundo y el sexto kernel de los Livermore Loops (LFK) [13]. Estos kernels explotan el paralelismo de grano fino con alta frecuencia de barreras [11].

La Fig. 6 muestra el código de Recurrence. Nosotros hemos añadido un chunk interno (líneas 7 a 12) para soportar diferentes tamaños de transacción. Primero, el kernel distribuye el cómputo entre hilos (líneas 1 a 3). El bucle de las líneas 5 a 14 actualiza un sólo elemento del array w accediendo a varios

```
1:  ochunk ← N/#TH
2:  start ← tid*ochunk
3:  stop ← start + ochunk
4:  for (t ← 0; t ≤ N-2; t ← t+1) do
5:    for (k ← start; k < stop; ) do
6:      TX_START(tx) ▷ Not necessary when using check_spec
7:      for (c ← 0; c < ichunk; c ← c+1) do
8:        if (k < (N-t-1)) then
9:          w[t+k+1] ← w[t+k+1] + b[k][t+k+1]*w[t]
10:       end if
11:       k ← k + 1
12:     end for
13:     TX_STOP(tx) / CHECK_SPEC(tx)
14:   end for
15:   SPEC_BARRIER(tx)
16: end for
17: LAST_BARRIER(tx)
```

Fig. 6. Livermore loop 6 (Recurrence).

```
1:  ii ← N
2:  ipntp ← 0
3:  repeat
4:    ipnt ← ipntp
5:    ipntp ← ipntp+ii
6:    ii ← ii/2
7:    i ← ipntp-1
8:    ochunk ← (ipntp-ipnt)/2 + (ipntp-ipnt) %2
9:    ochunk ← ochunk/#TH + ((ochunk %#TH)?1:0)
10:   i ← tid*ochunk
11:   end ← (ochunk*2*(tid+1)) + ipntp + 1
12:   start ← ipnt + 1 + (tid*2*ochunk)
13:   for (k←start; k < end; ) do
14:     TX_START(tx) ▷ Not necessary when using check_spec
15:     for (c ← 0; c < ichunk; c ← c+1) do
16:       i ← i+1
17:       x[i] ← x[k] - v[k]*x[k-1] - v[k+1]*x[k+1]
18:       k ← k + 2
19:     end for
20:     TX_STOP(tx) / CHECK_SPEC(tx)
21:   end for
22:   SPEC_BARRIER(tx)
23: until (ii > 1)
24: LAST_BARRIER(tx)
```

Fig. 7. Livermore loop 2 (Cholesky).

elementos de w y de la matriz b. Esto crea una recurrencia debido al patrón RAW en w, por lo que la barrera en la línea 15 es necesaria al final de cada iteración del bucle externo (línea 4 a 16). Sin embargo, las dependencias en w no se producen siempre, resultando un escenario prometedor para el uso de SBs. La LAST_BARRIER en la línea 17 asegura que no se libera memoria hasta que los hilos especulativos han acabado.

La Fig. 7 muestra la factorización incompleta de Cholesky basada en [11]. Se ha añadido un chunk interno (líneas 15 a 19) para soportar diferentes tamaños de transacción. El bucle interno (líneas 13 a 21) actualiza elementos de la mitad superior del array x (línea 17) con elementos de x y v. Una barrera en la línea 22 asegura que cada iteración del bucle externo (líneas 3 a 23) acceden a elementos actualizados de x. Sólo una fracción de elementos de x se actualiza en cada paso, siendo una aplicación adecuada para el uso de SBs. CHECK_SPEC se puede usar si estamos seguros de que no hay dependencias en el bucle interno, lo que no es trivial, aunque se dice que no hay dependencias en el código original [11].

DGCA [14] es un algoritmo de coloreado de grafos que no requiere información completa de todo el grafo para computar el color de cada nodo. Se mantiene

```
1:  p[N][#colors]                          ▷ Global color probability array
2:  start ← tid*N/#TH
3:  stop ← start + N/#TH
4:  for (m ← 0; t < M; m ← m+1) do
5:    for (i ← start; i < stop; i ← i + 1) do
6:      TX_START(tx) ▷ Not necessary when using check_spec
7:      collision = 0
8:      for (j ← 0; j < #adjacentNodes(i); j ← j+1) do
9:        if (nodeColor(i) = nodeColor(j)) then
10:         collision ← 1
11:       end if
12:     end for
13:     if collision then
14:       for (k ← 0; k < #colors; k ← k + 1) do
15:         p[i] ← updateNodeColorProbCollision(i,k)
16:       end for
17:     else
18:       for (k ← 0; k < #colors; k ← k + 1) do
19:         p[i] ← updateNodeColorProbNoCollision(i,k)
20:       end for
21:     end if
22:     updateNodeColor(i,p)      ▷ Potentially updates node
    color
23:     TX_STOP(tx) / CHECK_SPEC(tx)
24:   end for
25:   SPEC_BARRIER(tx)
26: end for
27: LAST_BARRIER(tx)
```

Fig. 8. Decentralised graph colouring algorithm (DGCA).

```
1:  [start,stop] ← createPartition(#edges, tid)
2:  #vert ← #vertices(startVertex,start,stop)
3:  xBegin()                               ▷ Standard transaction
4:  globMax#Vert ← max(globMax#Vert,#vert)
5:  xCommit()
6:  barrier()                              ▷ Standard barrier
7:  if (tid = 0) then
8:    outDegree ← dynamicAllocation(globMax#Vert)
9:    outVertexIndex ← dynamicAllocation(globMax#Vert)
10: end if
11: barrier()
12: [start,stop] ← createPartition(globMax#Vert,tid)
13: [outDegree,outVertexIndex] ← initialization(start,stop)
14: SPEC_BARRIER(tx)
15: outVertListSize ← fillArrays(outDegree,outVertexIndex)
16: LAST_BARRIER(tx)
17: prefixSums(outVertexIndex,outDegree,globMax#Vert) ▷
    Dynamic allocation and barriers inside
18: SPEC_BARRIER(tx)
19: TX_START(tx)
20: globOutVertListSize ← globOutVertListSize + out-
    VertListSize
21: TX_STOP(tx)
22: LAST_BARRIER(tx)
23: ...
```

Fig. 9. SSCA2 (Kernel 1).

un array global que mantiene la función de probabilidad de cada color para cada nodo y que se actualiza usando el color de los nodos vecinos. El código se muestra en la Fig 8. Las líneas 4 a 26 comprenden un paso de la ejecución donde se comprueba que un nodo no coincida en color con sus vecinos (líneas 7 a 12). Con esta información se actualiza p en las líneas 13 a 21, asignando una probabilidad máxima al color del nodo si no hay coincidencias, y actualizando la probabilidad del color de otra manera. Las dependencias se generan en la línea 22, donde el color del nodo se actualiza o no dependiendo de p. Puesto que sólo una fracción decreciente de nodos se cambia en cada paso, es un buen escenario para el uso de SBs.

SSCA2 [15] es una aplicación de teoría de grafos que consiste en cuatro kernels que muestran un acceso irregular a un multi-grafo dirigido. STAMP [16] incluye una versión paralela transaccional de SSCA2 y se centra en el kernel 1 debido a su adecuación a TM. Nosotros hemos usado SBs en el kernel 1 y el 4 que muestran un número considerable de barreras.

El kernel 1 construye un grafo a partir de una lista computando los arrays de adyacencia para los vértices de un grafo denso. La Fig. 9 muestra el código con barreras en las líneas 6 y 11 que no deberían reemplazarse por SBs ya que se utilizan instrucciones para la reserva de memoria de arrays que podrían ser usados por transacciones SB sin ser reservados. Hemos usado SBs en la línea 18 y dentro de la función *prefixSums()*. Este kernel no ofrece demasiadas oportunidades para el uso de SBs.

El kernel 4 de SSCA2 (Fig. 10) divide el grafo en clusters de nodos con alta conectividad y minimiza los enlaces entre ellos. El bucle en la línea 4 computa los clusters en paralelo. La barrera de la línea 8 podría ser tradicional ya que el código de después trabaja con los arrays que se computan antes, pero hemos usado una SB en su lugar. La barrera siguiente es adecuada para el uso de SBs ya que la única dependencia es con la inicialización de *globCliqueSize* por el hilo 0 en la línea 10. La siguiente barrera debería ser una LAST_BARRIER puesto que hay dependencias con *globCliqueSize* y *globIter*. Después del bucle, las listas parciales de cortes del grafo se unen para proceder con la minimización de enlaces entre clusters. Hemos dejado sin tocar la barrera de la línea 30 debido a la reserva de memoria, mientras que la barrera en la línea 33 se puede sustituir por una SB.

E. Directrices de uso de las SBs

Enumeramos aquí una serie de directrices para facilitar el uso eficiente de las SBs. Debemos examinar cada barrera del código y no reemplazarla por una SB si: (i) El código previo a la barrera reserva memoria que se usa después de la barrera. Acceder memoria sin reservar por medio de una transacción SB podría resultar en una violación de segmento o un aborto dependiendo del sistema HTM. (ii) El código posterior a la barrera ejecuta instrucciones que siempre abortarán una transacción, tales como llamadas al sistema, interrupciones,... Si colocamos una SB este caso sólo nos arriesgamos a que haya una disminución del rendimiento; (iii) Hay dependencias fáciles de detectar que evitarían que la especulación terminara.

Debemos reemplazar una barrera con una SB si: (i) Las dependencias no son evidentes y la barrera se ha colocado por precaución; (ii) Las dependencias varían dependiendo de los datos, e.g. algoritmos con dependencias sólo en los límites de las particiones de los datos.

En caso de que las barreras estén muy alejadas unas de otras se pueden colocar CHECK_SPECs para tener más puntos de comprobación de la especulación. Una buena práctica es medir el tiempo de la ejecución y utilizarlo como información para la colocación de SBs.

```
1:  ...
2:  vertVisited ← 0
3:  iter ← 0
4:  while (vertVisited < #vert ∨ iter < #vert/2) do
5:    if (tid = 0) then
6:      chooseVertToStart()
7:    end if
8:    SPEC_BARRIER(tx)
9:    if (tid = 0) then
10:     globCliqueSize ← 0
11:   end if
12:   [cliqueSize,cutSetIndex] ← determineClusters(iter)
13:   SPEC_BARRIER(tx)
14:   if (tid = 0) then
15:     iter ← iter + 1
16:     globIter ← iter
17:   end if
18:   TX_START(tx)
19:   globCliqueSize ← globCliqueSize + cliqueSize
20:   TX_STOP(tx)
21:   LAST_BARRIER(tx)
22:   iter ← globalIter
23:   vertVisited ← vertVisited + globCliqueSize
24: end while
25: barrier()                          ▷ Merge partial cutset lists
26: if (tid = 0) then
27:   edgeStartCount ← dynamicAllocation(#THs)
28:   edgeEndCount ← dynamicAllocation(#THs)
29: end if
30: barrier()
31: edgeEndCount[tid] ← cutSetIndex
32: edgeStartCount[tid] ← 0
33: SPEC_BARRIER(tx)
34: if (tid = 0) then
35:   for (t ← 1; t < #TH: t ← t + 1) do
36:     edgeEndCount[t] ← edgeEndCount[t] +
    edgeEndCount[t-1]
37:     edgeStartCount[t] ← edgeEndCount[t-1]
38:   end for
39: end if
40: TX_START(tx)
41: globCutSetIndex ← globCutSetIndex + cutSetIndex
42: TX_STOP(tx)
43: LAST_BARRIER(tx)
44: ...
```

Fig. 10. SSCA2 (Kernel 4).

III. Evaluación Experimental

Hemos utilizado un procesador IBM Power8 que incluye extensiones HTM best-effort [7] que proporcionan aislamiento fuerte, manejo de versiones lazy, detección de conflictos eager con granularidad de línea de caché, y una política de resolución de conflictos mixta donde las lecturas se resuelven con una política requester-loses y las escrituras con requester-wins. El procesador admite hasta 8KB de accesos transaccionales por núcleo. Las acciones escapadas se implementan por medio del *suspended-mode* que se puede activar con las instrucciones *tsuspend/tresume*. La evaluación se realizó en un servidor S822LC-8335 con 2 sockets y procesadores de 10 núcleos. Cada núcleo puede ejecutar hasta 8 hilos SMT resultando un total de 160 hilos de ejecución.

A. Metodología

Las métricas analizadas incluyen el *speedup* con respecto a la ejecución secuencial no transaccional, el *transaction commit ratio* (TCR), que es el porcentaje de las transacciones que hacen commit con respecto al número total de transacciones iniciadas, y el *speculative commit ratio* (SCR), que es el porcentaje de transacciones SB que hacen commit con respecto a todas las SB iniciadas. Ejecutamos 20 veces cada experimento y obtenemos la media del tiempo de ejecución. Para reducir la variabilidad atamos cada hilo a un núcleo para prevenir migraciones. Usamos 128 núcleos del servidor con la afinidad hilo/núcleo distribuida con un esquema round-robin evitando el SMT en lo posible, lo que da lugar a tres escenarios diferentes: (i) de 1 a 8 hilos sólo se usa un socket por lo que se reduce el impacto de la comunicación entre sockets. En este caso sólo se usa un núcleo por hilo por lo que se aprovechan todos los recursos transaccionales; (ii) los experimentos con 16 hilos utilizan 2 sockets pero sin SMT; (iii) de 32 hilos en adelante se utilizan los dos sockets con SMT por lo que tenemos la latencia de la comunicación entre sockets y los recursos transaccionales repartidos entre hilos SMT. Usamos *padding* para fijar las variables de las líneas 4 y 5 de la Fig. 2 en bloques de caché diferentes y así prevenir conflictos.

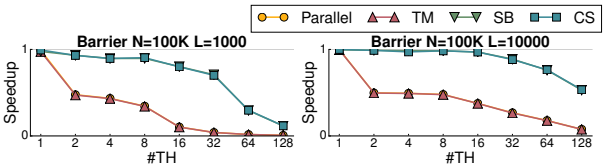


Fig. 11. Resultados del microbenchmark Barrier.

Hemos considerado 5 perfiles diferentes para la evaluación de cada aplicación:

- *Unprotected*: no usa ni transacciones ni barreras, por lo que los resultados pueden ser incorrectos. Este perfil representa un limite superior del rendimiento que se puede llegar a conseguir.
- *Parallel*: con las barreras tradicionales originales de los códigos y sin transacciones ni SBs.
- *CS*: similar a Parallel, pero con SBs en lugar de las tradicionales y con la primitiva CHECK_SPEC para añadir comprobaciones de la especulación.
- *TM*: con transacciones y las barreras tradicionales originales de los benchmarks.
- *SB*: similar a TM, pero con SBs. En este caso las transacciones entre barreras ya proporcionan las comprobaciones de la especulación, por lo que no es necesario el uso de CHECK_SPEC.

B. Resultados

Microbenchmark Barrier

La Fig. 11 muestra los resultados obtenidos para el microbenchmark Barrier de la Fig. 5. El eje y es el speedup sobre la aplicación secuencial, que en este caso particular representa la eficiencia, ya que el benchmark está implementado de manera que la carga no se reparte entre los hilos sino que se replica. De esta manera se puede medir la perdida de rendimiento en barreras y transacciones.

La espera en barrera producida por el desbalanceo de la carga que tiene el benchmark (líneas 3 a 8 en Fig. 5) se puede notar en los perfiles sin SBs (Parallel y TM) donde la eficiencia cae a la mitad. Sin embargo, SB y CS pueden especular más allá de la barrera de manera que los hilos ociosos en una

iteración pueden avanzar y empezar el trabajo de la siguiente. En la gráfica de la izquierda los hilos realizan menos trabajo (L) y se puede ver cómo afecta la contención de barrera, sobre todo cuando se usan los 2 sockets, de 16 hilos en adelante. A los perfiles Parallel y TM les afecta tanto el desbalanceo como la contención de barrera, y los SB y CS también empeoran su rendimiento pero debido sólo a la contención a partir de 16 hilos. Todos los experimentos tienen un 100 % de SCR con SB y CS, como se muestra en la Tabla I, confirmando que el API no introduce falsos conflictos.

TABLA I
TCR y SCR para 16 hilos (%).

Bench		TCR/SCR		
		TM	SB	CS
Barr	L1K	99.99/-	99.99/99.99	-/99.98
	L10K	99.96/-	99.75/99.97	-/99.95
Recurren	C1	98.63/-	99.30/6.37	-/34.60
	C5	95.56/-	98.64/8.31	-/33.09
	C10	92.18/-	97.91/8.42	-/26.68
	C15	88.64/-	96.89/7.91	-/29.20
Cholesky	C1	100.00/-	100.00/38.04	-/60.87
	C5	99.98/-	100.00/43.82	-/58.83
	C10	99.97/-	100.00/45.14	-/59.25
	C15	99.95/-	100.00/45.71	-/57.08
DGCA	N2K A2	99.78/-	99.92/99.60	-/99.00
	N8K A2	99.95/-	99.94/99.36	-/99.76
	N2K A8	94.24/-	93.29/42.04	-/48.59
	N8K A8	98.13/-	97.79/29.59	-/40.31
SSCA2	k1-s14	98.64/-	98.54/3.04	-
	k1-s20	99.98/-	99.97/0.18	-
	k4-s14	71.77/-	30.07/0.51	-
	k4-s20	62.61/-	30.47/0.05	-

Recurrence

La Fig. 12 muestra los resultados para Recurrence. Hemos variado el tamaño del chunk (C) de 1 a 15 iteraciones. Se pueden observar dos comportamientos: los perfiles Unprotected, Parallel y CS no muestran la sobrecarga debido a transacciones, mientras que TM y SB sí. Esto se puede apreciar cuando se usan de 1 a 8 hilos, donde los primeros obtienen speedups similares y los segundos también pero más bajos. A partir de 16 hilos se aprecia un comportamiento superlineal con Unprotected y CS debido a la caché. Recurrence tiene poco reuso de datos en su ejecución secuencial debido al patrón irregular de accesos a la matriz b (Fig. 6, línea 9), y al usar más hilos se disminuye el número de fallos de caché hasta 10× por el uso de las cachés privadas de los núcleos. Por otro lado, se puede observar que el perfil Parallel no escala a partir de 16 hilos debido a la contención de barrera cuando se usan 2 sockets. Esto se puede ver en el perfil TM también, mientras que la especulación de barrera en el perfil SB reduce esta sobrecarga. A partir de 32 hilos se nota la sobrecarga sobre los recursos transaccionales debido al SMT en los perfiles que usan transacciones, sobre todo con chunks más grandes que implican transacciones mayores.

La Tabla I muestra el TCR y SCR para 16 hilos. Para Recurrence los resultados de TCR son buenos. El incremento del chunk produce ligeros decrementos en los porcentajes. Sin embargo, CS produce mejores porcentajes de SCR, lo que es lógico debido a que hay menos transacciones con las que entrar en conflicto (no están las transacciones entre barreras).

Cholesky

Los resultados para Cholesky se muestran en la segunda fila de la Fig. 12. Este benchmark presenta dos diferencias importantes con respecto a Recurrence. Primero, las dependencias son menos frecuentes, lo que se refleja en mejores valores de TCR y SCR, y la mejor escalabilidad independientemente del perfil. Y segundo, la superlinealidad no es tan evidente, debido a que los fallos de caché se reducen hasta 0,75× en versiones paralelas.

El rendimiento del perfil CS se encuentran entre el Unprotected y el Parallel independientemente del número de hilos. Nótese que la diferencia de rendimiento entre estos perfiles se incrementa a partir de 16 hilos debido a la contención en barreras cuando se usan 2 sockets. Este efecto también está presente cuando se comparan los perfiles TM y SB. En este caso, las diferencias son más evidentes a partir de 32 hilos. La sobrecarga causada por las transacciones enmascara las diferencias de rendimiento.

DGCA

Los resultados de DGCA se muestran en la tercera fila de la Fig. 12. Los gráficos muestran diferentes configuraciones variando el grado de adyacencia de los nodos (A) y el número de nodos (N) del grafo. Las conexiones individuales entre nodos se eligen aleatoriamente siguiendo una distribución uniforme. De esta manera, obtenemos una variedad de configuraciones con escenarios de baja contención con grafos pequeños y grandes en las dos primeras gráficas, y escenarios de alta contención en el resto.

En los escenarios de baja contención se obtiene una sobrecarga notable en los perfiles que usan transacciones hasta 8 hilos. En este caso, los perfiles Unprotected y Parallel tienen mejor rendimiento que CS, mientras que TM y SB son parecidos. Sin embargo, Parallel se ve afectado por la contención en barreras de 8 hilos en adelante, obteniendo speedups comparables a TM. La especulación funciona en estos escenarios con CS y SB escalando hasta los 32 hilos y obteniendo los valores más altos de SCR, como se puede ver en la Tabla I. Con el grafo más grande (N=8000), en la segunda gráfica podemos ver un comportamiento similar con ganancias en perfiles especulativos hasta los 64 hilos debido a la menor probabilidad de conflicto entre transacciones.

La escalabilidad en los escenarios de alta contención (las dos gráficas de la derecha) se ve limitada principalmente por las dependencias. En estos experimentos las diferencias entre CS, Parallel y Unprotected se reducen con 1-8 hilos debido a la ejecución más intensiva en cómputo asociada con una mayor conectividad. Nótese la diferencia de rendimiento entre el perfil CS y el SB a partir de 32 hilos en la gráfica de la derecha. En este caso SB baja mucho

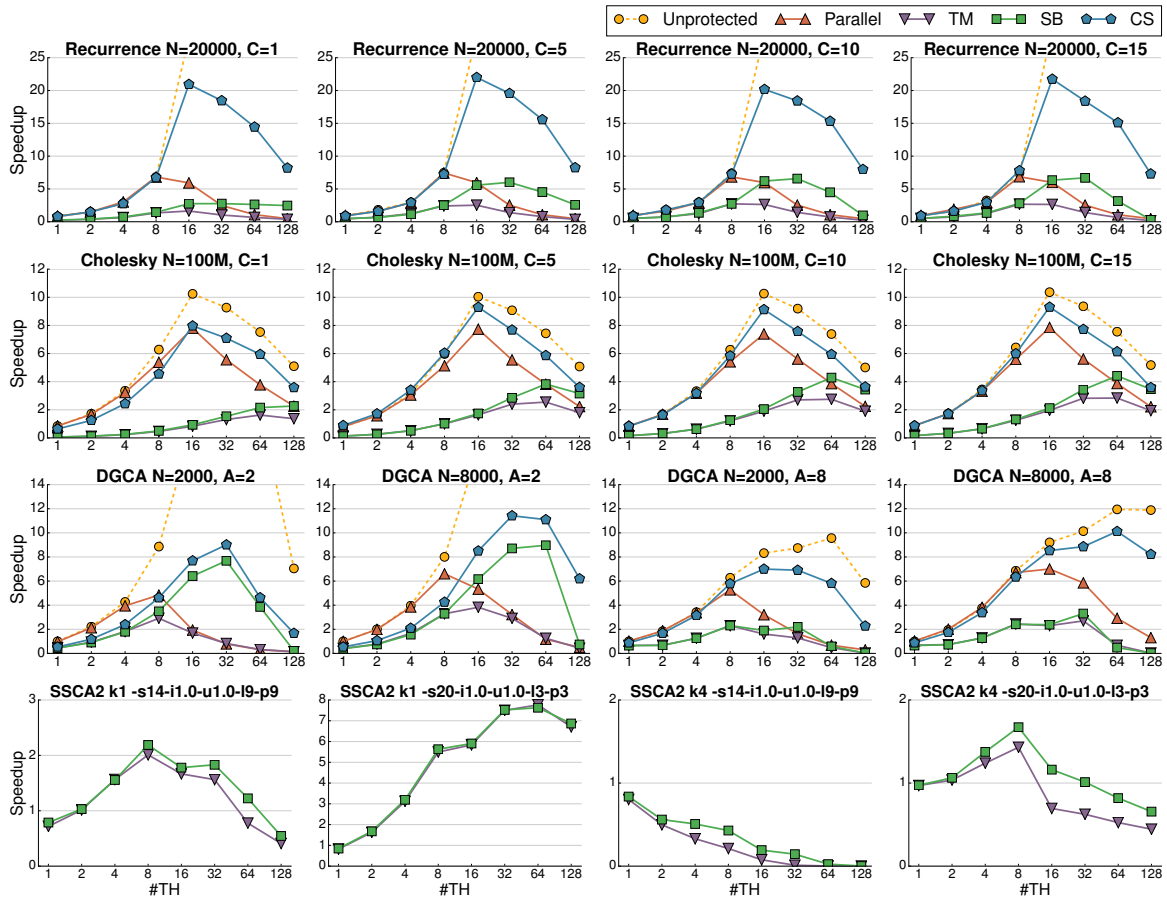


Fig. 12. Resultados de Power8 para Recurrence (Livermore loop 6), Cholesky (Livermore loop 2), DGCA and SSCA2.

su rendimiento ya que las transacciones normales y las SB comparten recursos transaccionales, y el uso de SMT causa abortos por capacidad. Esto no ocurre con CS porque no hay transacciones normales, ni con A=2, ya que las transacciones acceden a menos posiciones de memoria por la menor conectividad.

Las diferencias entra las distintas configuraciones son especialmente evidentes en el TCR y el SCR en la Tabla I. Se aprecia un alto TCR en todos los experimentos mientras que el SCR depende del tamaño del grafo y el número de conexiones, que determinan los potenciales conflictos.

SSCA2

Los resultados para SSCA2 se muestran al final de la Fig. 12. Como parte de STAMP estos benchmarks ya están paralelizados usando transacciones por lo que sólo se consideran los perfiles SB y TM.

El kernel 1 muestra ganancias limitadas de SB sobre TM en la primera gráfica y ninguna en la segunda que tiene una entrada más grande. La mayoría de las barreras del kernel protegen reserva dinámica de memoria e impiden el uso de SBs. Además, a diferencia de los benchmarks previos, las barreras no aparecen dentro de bucles, y como consecuencia sólo un centenar de barreras se llaman a lo largo del código reduciendo las oportunidades de especulación.

En el kernel 4 las principales oportunidades de especulación están en el bucle while de las líneas 4 a 24 de la Fig. 10, que contiene dos SBs. La primera, en la línea 8, especula sobre la función *determineClusters* (línea 12) que, al igual que *fillArrays* del kernel 1, contienen bucles que causan abortos por capacidad. Sin embargo, la computación de *determineClusters* se decrementa con las iteraciones del bucle, por lo que alguna transacción SB puede llegar a acometer sus cambios.

Desafortunadamente, los conflictos causados por la operación de reducción de la línea 19 y las frecuentes interacciones con variables globales en las líneas 10, 16, 22 y 23 producen abortos que limitan el rendimiento. Esto se traduce en los pequeños speedups de la Fig. 12. Este hecho se puede observar también en los bajos valores de TCR y SCR de la Tabla I.

IV. Trabajo Relacionado

La idea de usar técnicas de especulación con código heredado para facilitar la explotación de las arquitecturas multicore no es nueva. Algunas técnicas se centran en la especulación de códigos paralelos donde se consideran las iteraciones de bucles y las llamadas a funciones como objetivos principales de la especulación. En [17] se identifican las limitaciones de TM y encuentran que existe una falta de soporte para diferenciar hilos más o menos especulativos, la falta de forwarding o los conflictos por false sharing como puntos a mejorar del soporte especulativo.

Torrellas et. al. introducen *Speculative Synchronization* como una técnica que aplica Thread-Level Speculation (TLS) a código paralelo. En [18] proponen soporte hardware para permitir especulación en locks, flags y barreras. El hardware comprueba de-

pendencias entre hilos y descarta el trabajo especulativo en caso de conflicto. Esta propuesta no soporta restricciones de orden y no resuelve dependencias de nombre, lo que simplifica los requisitos del hardware.

Otras propuestas se orientan a implementar TLS con extensiones HTM. En [19] se analizan las limitaciones que muestra Intel TSX para la especulación comparado con otras propuestas que aplican hardware adicional no presente en procesadores reales. Se centran en bucles de aplicaciones SPEC CPU2006 y modifican el código manualmente para soportar la especulación. El rendimiento se degrada en la mayoría de los casos (TSX no ofrece acciones escapadas).

En cuanto a optimización de barreras, en [20] se analizan los efectos de la sincronización con barreras de grano fino. Los autores encuentran una cifra relativamente baja de dependencias y proponen un esquema de optimización de código para asumir que no hay dependencias entre dos barreras consecutivas. Usan la Advance Load Address Table (ALAT) presente en los procesadores Itanium para detectar posibles fallos en la especulación en tiempo real.

En [12] se usa soporte HTM comercial para especular en barreras. Esta propuesta requiere especificar manualmente un punto de sincronización para los hilos especulativos después de la barrera. No se proporciona soporte para poner restricciones de orden a las transacciones ni se utilizan acciones de escape, por lo que no hay mejora de rendimiento. Tampoco tienen en cuenta las limitaciones del sistema HTM al no proporcionar un control de la longitud de la especulación.

V. Conclusiones

En este artículo se propone una barrera especulativa (SB) optimista como alternativa a las barreras tradicionales basadas en locks. Las SBs permiten que los hilos salten la barrera usando ejecución especulativa con HTM. Proporcionamos un API para las SBs implementado con extensiones transaccionales para su uso en aplicaciones con y sin transacciones.

Evaluamos nuestras propuestas usando un servidor IBM Power8 que proporciona las extensiones transaccionales necesarias para nuestra propuesta ya que ofrece la posibilidad de ejecutar acciones escapadas dentro de transacción. Los resultados muestran una mejora general del rendimiento sobre las barreras tradicionales con speedups de hasta 6× en media sobre la aplicación paralela para ciertas configuraciones. La configuración en 2 sockets del servidor IBM penaliza la comunicación de las barreras tradicionales mientras que las SBs ocultan la latencia de la contención de barrera. La sobrecarga de las SBs no parece afectar el rendimiento negativamente en la mayoría de benchmarks evaluados. En general se recomienda el uso de SBs incluso cuando hay dependencias obvias entre barreras.

Agradecimientos

Este trabajo ha sido posible gracias a los proyectos TIN2016-80920-R y P12-TIC-1470.

Referencias

[1] M. Herlihy and J.E.B. Moss, "Transactional memory: Architectural support for lock-free data structures," in *Int'l. Symp. on Computer Architecture (ISCA'93)*, 1993, pp. 289–300.

[2] L. Hammond, V. Wong, M. Chen, et al., "Transactional memory coherence and consistency," in *Int'l. Symp. on Computer Architecture (ISCA'04)*, 2004, pp. 102–113.

[3] R. Rajwar, M. Herlihy, and K. Lai, "Virtualizing transactional memory," in *Int'l. Symp. on Computer Architecture (ISCA'05)*, 2005, pp. 494–505.

[4] K.E. Moore, J. Bobba, Moravan, et al., "LogTM: Log-based transactional memory," in *Int'l. Symp. on High-Performance Computer Architecture (HPCA'06)*, 2006, pp. 254–265.

[5] C. Kevin Shum, Fadi Busaba, and Christian Jacobi, "IBM zEC12: The third-generation high-frequency mainframe microprocessor," *IEEE Micro*, vol. 33, no. 2, pp. 38–47, 2013.

[6] P. Hammarlund and A. J. Martinez et. al., "Haswell: The fourth-generation intel core processor," *IEEE Micro*, vol. 34, no. 2, pp. 6–20, 2014.

[7] H. Q. Le, G. L. Guthrie, D. E. Williams, et al., "Transactional memory support in the IBM POWER8 processor," *IBM Journal of Research and Development*, vol. 59, no. 1, pp. 8:1–8:14, Jan 2015.

[8] Ricardo Quislant, Eladio Gutierrez, Emilio L Zapata, and Oscar Plata, "Insights into the Fallback Path of Best-Effort Hardware Transactional Memory Systems," in *Int'l. Conf. on Parallel Processing (Euro-Par'16)*, 2016, pp. 251–263.

[9] Michelle J Moravan, Jayaram Bobba, Kevin E Moore, et al., "Supporting Nested Transactional Memory in logTM," in *Int'l. Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS'06)*, 2006, pp. 359–370.

[10] Manuel Pedrero, Eladio Gutierrez, and Oscar Plata, "TMbarrier: Speculative Barriers Using Hardware Transactional Memory," in *Euromicro Int'l. Conf. on Parallel, Distributed and Network-Based Processing (PDP'18)*, 2018, pp. 214–221.

[11] J Sampson, R Gonzalez, J Collard, et al., "Exploiting Fine-Grained Data Parallelism with Chip Multiprocessors and Fast Barriers," in *Int'l. Symp. on Microarchitecture (MICRO'06)*, 2006, pp. 235–246.

[12] Lars Bonnichsen and Artur Podobas, "Using Transactional Memory to Avoid Blocking in OpenMP Synchronization Directives," in *Int'l. Workshop on OpenMP (IWOMP'15)*, 2015, pp. 149–161.

[13] John T. Feo, "An analysis of the computational and parallel complexity of the Livermore Loops," *Parallel Computing*, vol. 7, no. 2, pp. 163–185, 1988.

[14] K R Duffy, N O'Connell, and A Sapozhnikov, "Complexity analysis of a decentralised graph colouring algorithm," *Information Processing Letters*, vol. 107, no. 2, pp. 60–63, 2008.

[15] David A Bader and Kamesh Madduri, "Design and Implementation of the HPCS Graph Analysis Benchmark on Symmetric Multiprocessors," in *Int'l. Conf. on High Performance Computing (HiPC'05)*, 2005, pp. 465–476.

[16] C.C. Minh, J. Chung, C. Kozyrakis, and K. Olukotun, "STAMP: Stanford Transactional Applications for Multi-Processing," in *Int'l. Symp. on Workload Characterization (IISWC'08)*, 2008, pp. 35–46.

[17] Leo Porter, Bumyong Choi, and Dean M. Tullsen, "Mapping out a path from hardware transactional memory to speculative multithreading," in *Int'l. Conf. on Parallel Architectures and Compilation Techniques (PACT'09)*, 2009, pp. 313–324.

[18] José F. Martínez and Josep Torrellas, "Speculative synchronization: Applying thread-level speculation to explicitly parallel applications," *SIGOPS Oper. Syst. Rev.*, vol. 36, no. 5, pp. 18–29, 2002.

[19] R. Odaira and T. Nakaike, "Thread-level speculation on off-the-self hardware transactional memory," in *Int'l. Symp. on Workload Characterization (IISWC'14)*, 2014, pp. 212–221.

[20] Vijay Nagarajan and Rajiv Gupta, "Speculative optimizations for parallel programs on multicores," in *Int'l. Workshop on Languages and Compilers for Parallel Computing (LCPC'09)*, 2009, pp. 323–337.

CNN-Sim: Un Simulador de Arquitecturas para Procesamiento de Redes Neuronales Convolucionales

Francisco Muñoz-Martínez[1], José L. Abellán[2] y Manuel E. Acacio[1]

Resumen— **Una Red Neuronal Convolucional (CNN) es un modelo de inteligencia artificial (IA) que trata de emular el funcionamiento del cerebro para resolver tareas complejas tales como la clasificación de imágenes, la conducción autónoma, o la detección de objetos. Hoy en día incluso están siendo utilizadas para la detección de distintos de tipos cáncer o la traducción del lenguaje natural.**

Dado que la aplicación de las CNNs requiere normalmente de procesamiento en tiempo real, numerosas plataformas específicas de cómputo han surgido para su ejecución. Estas plataformas, denominadas aceleradores, están co-diseñadas con las características computacionales de las CNN de manera que permiten maximizar la reutilización de datos mediante la implementación de un flujo de datos o *dataflow*, e incluso integran optimizaciones para evitar cálculos innecesarios como las multiplicaciones por cero, o la reutilización de operandos, entre otras técnicas. Gracias a esto, los aceleradores actuales consiguen una eficiencia energética muy superior a la obtenida por una GPU o una CPU tradicional en el procesamiento de estas cargas de trabajo. Es por ello por lo que la mejora de los sistemas aceleradores actuales se torna esencial para garantizar una evolución continua de las aplicaciones de IA que demanda nuestra sociedad. El problema fundamental es que la mayoría de estos diseños de acelerador son propietarios y, por tanto, sus detalles permanecen ocultos a la comunidad científica en general, lo que limita enormemente la evaluación de nuevas propuestas que puedan mejorar los sistemas actuales.

En este trabajo presentamos *CNN-Sim*, el primer simulador de CNNs de código abierto con precisión a nivel de ciclo que implementa y permite evaluar dataflows bien conocidos tales como el *Weight Stationary* y *Output Stationary*. Esta herramienta, escrita en el lenguaje de programación Python y conectada con el framework de CNNs Caffe, permite la exploración de técnicas arquitecturales para optimizar el procedimiento de inferencia en aceleradores hardware específicos para CNNs. Para demostrar el potencial de *CNN-Sim* mostramos por primera vez la influencia del tráfico de red inservible (ceros) en ambos dataflows. En concreto, obtenemos que el porcentaje de ceros en la red de interconexión constituye entre un 31 % y un 34 % del tráfico de red total.

Palabras clave— **Red Neuronal Convolucional (CNN), AlexNet, Simulación, SCALE-Sim.**

I. Introducción

LA s Redes Neuronales Profundas (*Deep Neural Networks* o DNNs) constituyen hoy en día la base de muchas aplicaciones modernas de inteligencia artificial [1]. Desde su aplicación con éxito a los campos del reconocimiento de imágenes [2] y reconocimiento del habla [3], las DNNs han sido empleadas en una gran variedad de contextos, que van desde vehículos sin conductor [4] hasta sistemas de detección de distintos tipos de cáncer [5], pasando por el mundo de los juegos complejos [6], llegando incluso a superar en alguno de ellos la capacidad humana.

Una red neuronal (NN) trata de emular la funcionalidad del cerebro a través de una colección de neuronas simples artificiales que se agrupan en una secuencia de capas, y las salidas de una capa constituyen las entradas de la siguiente. Cada neurona artificial aplica una función no-lineal, como por ejemplo *max(0, valor)*, al resultado de la suma ponderada de sus entradas utilizando para ello una serie de pesos (o *weights*) únicos a cada enlace de entrada de la neurona. La gran ventaja de las NNs sobre otras técnicas anteriores basadas en el uso de reglas diseñadas por expertos o características definidas a mano, es su capacidad para extraer características de alto nivel directamente desde los datos de entrada. Para ello las NNs emplean una representación del espacio de las entradas que se obtiene realizando un proceso de aprendizaje estadístico a partir de una gran cantidad de datos. El uso de la palabra *Profunda (deep)* aplicada a una NN viene de la posibilidad de tener más de unas pocas capas y cada una de ellas con un número de neuronas grande, como una forma de minimizar el error de predicción.

Esta mayor capacidad de las DNNs conlleva sin embargo una complejidad computacional grande que requiere de la utilización de dispositivos con una alta capacidad de procesamiento. Aunque las arquitecturas GPGPU han sido y seguramente seguirán siendo un pilar fundamental en el procesamiento de DNNs, cada vez van apareciendo más aceleradores especializados capaces de dar soporte eficiente a estas cargas de trabajo.

Estos aceleradores se basan en arquitecturas sistólicas, las cuales están formadas por un array bidimensional de elementos de procesamiento (*Processing Elements* o PEs) que se envían datos entre sí, reduciendo la cantidad de accesos a memoria principal necesarios y mejorando así la eficiencia energética respecto a una arquitectura tradicional.

La mayoría de estas arquitecturas específicas se han centrado en un tipo de DNN concreto denominado Red Neuronal Convolucional [7] (*Convolutional Neural Network* o CNN), entre las cuales destacan Eyeriss [8], DaDianNao [9], DC-CNN [10], FlexFlow [11] o MAERI [12].

La principal dificultad radica en que hasta el momento ninguno de estos aceleradores es liberado pa-

[1]Dpto. de Ingeniería y Tecnología de Computadores, Univ. Murcia, e-mail: {francisco.munoz2, meacacio}@um.es.

[2]Dpto. Grado en Ingeniería Informática, Univ. Católica San Antonio de Murcia, e-mail: jlabellan@ucam.edu.

ra ser evaluado por la comunidad científica, haciendo muy difícil una comparación fidedigna entre las distintas arquitecturas que implementan o entre los nuevos diseños que van siendo propuestos a lo largo del tiempo. Recientemente, ha surgido *SCALE-Sim* [13] como la primera aproximación de simulador de este tipo de arquitecturas. Sin embargo, esta herramienta presenta una gran cantidad de inconvenientes que la hacen inadecuada para poder llevar a cabo una simulación realista.

Con el objetivo de solucionar este problema, en este trabajo presentamos *CNN-Sim*, una versión extendida de *SCALE-Sim* que realmente simula ciclo a ciclo la ejecución de una CNN. Esta herramienta, conectada con el conocido framework de deep learning Caffe, es capaz de cargar los datos de entrada (activaciones y pesos) en una memoria simulada, y ejecutar de forma realista el procesamiento de la CNN imitando a un acelerador específico. Una vez la ejecución ha sido completada, el simulador devuelve a Caffe los datos de salida y este puede seguir su ejecución normalmente. Además, el simulador detalla el valor y la dirección de cada uno de los operandos que se van leyendo y escribiendo durante su ejecución, así como los ciclos que emplea y los accesos a memoria que se realizan.

Además, como caso de aplicación del simulador, en este trabajo se muestra un estudio acerca de la cantidad de ceros que fluyen por la red de interconexión del acelerador. Aunque la mayoría de propuestas actuales tratan de evitar estos ceros para mejorar el rendimiento y ahorrar energía en su arquitectura [8], generalmente estos valores son enviados a través de la red que interconecta los PEs, y son evitados únicamente cuando llegan a la unidad de cómputo. En este trabajo, usando nuestra herramienta *CNN-Sim*, mostramos que de media un 30 % de todo el tráfico de la red de interconexión podría ser evitado ya que son valores igual a cero que finalmente van a ser inútiles.

El resto del artículo se organiza de la siguiente manera: la sección II describe el funcionamiento de una CNN tradicional y la arquitectura general de un acelerador específico para CNNs. La sección III describe nuestro simulador *CNN-Sim* y los inconvenientes de utilizar la versión previa *SCALE-Sim*. La sección IV muestra la metodología utilizada en nuestro caso de estudio y los resultados obtenidos. Finalmente, en la sección V se muestran las conclusiones principales extraídas y el trabajo futuro.

II. Background

A. CNN tradicional

En la actualidad, las CNNs se han convertido en el tipo de red neuronal profunda (DNN) más importante en diferentes ámbitos, destacando mayormente el campo de la visión por computador. Podemos llamar CNN a cualquier red neuronal que tenga al menos una capa convolucional en su interior. En este trabajo, nos vamos a centrar en simular el procedimiento de inferencia de las redes CNN, procedimiento que se ejecuta una vez la red ha sido entrenada y que ocurre cuando ésta es realmente utilizada (por ejemplo, para predecir objetos en imágenes cuando la CNN es desplegada en una cámara a bordo de un vehículo con conducción autónoma).

El funcionamiento de una capa convolucional se ilustra en la Figura 1. Vemos que dicha capa recibe un conjunto de C fmaps (input fmaps o ifmaps) y obtiene un conjunto de N fmaps (output fmaps o ofmaps). Los ifmaps son simplemente arrays de datos de tamaño $H \times W$. Durante el desarrollo de este trabajo denominaremos a los datos dentro de un fmap como `píxeles`. Además, cada capa tiene un total de N conjuntos de filtros de pesos cuyos valores se obtienen tras la fase de entrenamiento de la NN. Cada conjunto está formado por C filtros de pesos de tamaño $R \times S$. Para generar cada ofmap específico de dimensiones $OH \times OW$ primero se realizan las convoluciones de todos los ifmaps C, con el conjunto de C filtros de pesos, asociado a dicho ofmap. Estas C convoluciones, generan C fmaps intermedios que son sumados para obtener el ofmap correspondiente. Finalmente, para obtener más cantidad de ofmaps (N), bastará con realizar el mismo proceso con conjuntos de pesos diferentes. Por esta razón tenemos una cantidad de conjuntos de filtros de pesos igual al número de ofmaps a obtener: N.

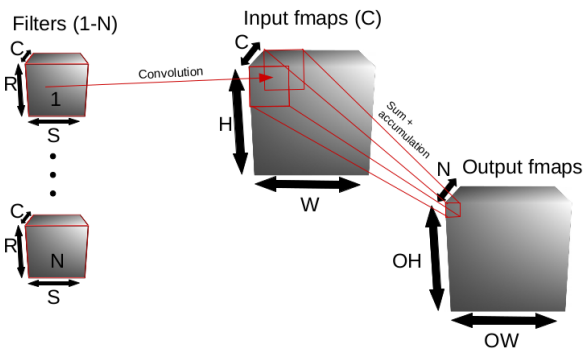


Fig. 1: Ejemplo de funcionamiento de una capa convolucional.

Además de estas capas convolucionales, una CNN suele incluir una o varias capas de *pooling*, las cuales, al igual que las anteriores, reciben fmaps de entrada y obtienen uno o varios fmaps de salida, aunque su funcionamiento es muy diferente y consiste en reducir los datos de entrada realizando alguna operación básica sobre los mismos, como por ejemplo el valor medio. Así, una capa de pooling, extrae las características más importantes de los datos de entrada con el objetivo de reducir el tamaño de los datos de las capas siguientes, reduciendo de esta forma el coste computacional y de almacenamiento de las mismas.

Finalmente, una CNN contiene al final una o varias capas *Fully-Connected* (FC) que se encargan, a partir de la extracción de características realizada durante todo el proceso anterior, de obtener el resultado final, como puede ser un vector de valores indicando

la probabilidad de clasificación de una imagen con respecto a las etiquetas asociadas de entrada.

Como podemos comprobar, el problema fundamental de este tipo de redes es el coste computacional de ejecutar estas capas convolucionales, que puede llegar a alcanzar un número significativo en CNNs actuales (107 capas en ResNet-110 [14]). Además, el uso de la memoria también es significativo pues es necesario almacenar tantos conjuntos de pesos como ofmaps queramos obtener (ResNet-110 consta de 55M de pesos).

Uno de los ejemplos más conocidos de este tipo de CNNs es AlexNet [7], cuyo modelo es presentado en la Tabla I y será utilizado en nuestro caso de estudio de la Sección IV. La columna 1 muestra el identificador de la capa. A partir de ahora nos referiremos a dicho identificador cuando queramos hacer referencia a cierta capa. La columna 2 muestra el nombre de la capa y las columnas 3 y 4 muestran las dimensiones de los filtros y el tamaño de los fmaps de entrada, respectivamente. Como vemos, AlexNet está formada por 5 capas convolucionales (IDs 0-4) y 4 capas Fully-Connected (IDs 5-7) cada una seguida de una capa ReLU [15], que se encarga de añadir no linearidad a la CNN y no es mostrada por simplicidad. Además, después de las capas con IDs 0,1 y 4 se encuentran 3 capas max-pooling que se encargan de reducir el tamaño de los ifmaps en las capas sucesivas.

TABLA I: Características de las capas de AlexNet.

ID	Type	Filter Shape	Input Size
0	conv1	$96 \times 3 \times 11 \times 11$	$3 \times 227 \times 227$
1	Conv2	$256 \times 48 \times 5 \times 5$	$48 \times 55 \times 55$
2	Conv3	$384 \times 256 \times 3 \times 3$	$256 \times 27 \times 27$
3	Conv4	$384 \times 192 \times 3 \times 3$	$192 \times 13 \times 13$
4	Conv5	$256 \times 192 \times 3 \times 3$	$192 \times 13 \times 13$
5	FC1	-	9216×4096
6	FC2	-	4096×4096
7	FC3	-	4096×1000

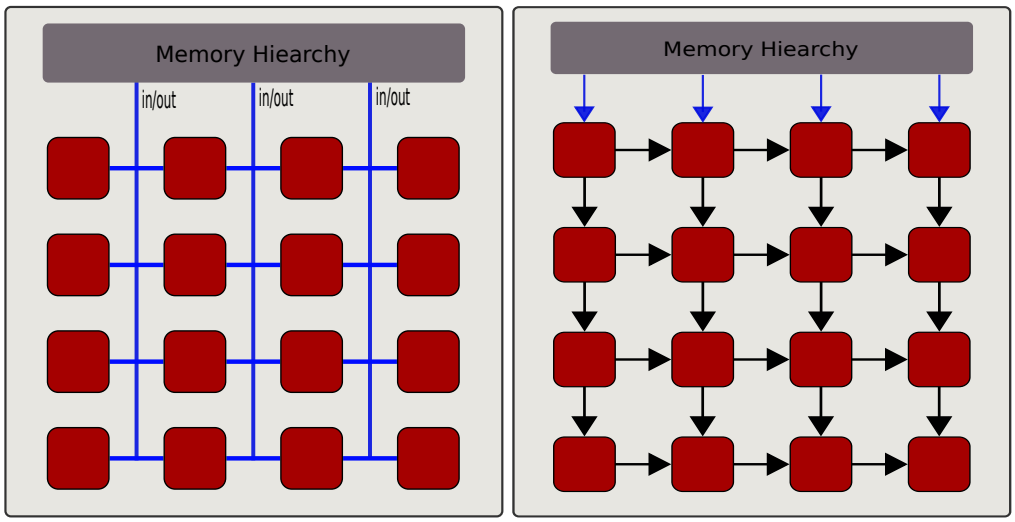
B. Arquitecturas específicas para CNNs

Tal y como acabamos de ver, la ejecución del procedimiento de inferencia en una CNN es intensivo tanto en cómputo como en uso de memoria, ya que requiere billones de cálculos de multiplicación y suma (MACs) y millones de accesos a memoria [16]. Esta es la razón por la cual una CPU o una GPU tradicional no puede obtener un rendimiento eficiente para este tipo de cargas de trabajo. Este tipo de arquitecturas tradicionales, cuyo esquema se muestra en la Figura 2a, consisten en varias unidades de procesamiento de propósito general, cada una de las cuales accede a memoria principal de forma independiente, generando un cuello de botella en la misma y en la red de interconexión del sistema. Además, concretamente, una CPU no puede aprovechar el alto grado de paralelismo que se encuentra en la ejecución ya que no tiene las suficientes unidades de procesamiento y además requiere de mecanismos de sincronización de hilos que sobrecargan el sistema en un tipo de paralelismo de grano tan fino. Por otro lado, aunque las GPUs poseen la capacidad de explotar este alto grado de paralelismo obteniendo un rendimiento adecuado, el problema fundamental de estas arquitecturas se encuentra en su alto consumo energético provocado en general por la significativa cantidad de accesos a memoria que se generan [17]. En la mayoría de los casos, el uso de estos dispositivos es inadecuado si lo que queremos es desplegar una CNN en el contexto de un sistema empotrado con capacidades limitadas de cómputo, de memoria y que funciona con una batería limitada [18], [19], [20], [21].

Por ello, en los últimos años, están surgiendo diferentes tipos de aceleradores que tratan de optimizar al máximo el procedimiento de inferencia de las CNNs. La mayoría de estos se basan en arquitecturas sistólicas (Figura 2b), las cuales están conformadas por cientos de elementos de procesamiento (PEs) capaces de ejecutar operaciones MAC, todos ellos conectados entre sí enviándose y recibiendo información, reutilizando datos entre ellos y por tanto disminuyendo de forma significativa los accesos a memoria principal, mejorando así la eficiencia energética del sistema.

Actualmente, existe una gran cantidad de arquitecturas aceleradoras sistólicas [8], [9], [10], [11], [22], [12], cada una de las cuales difiere en el flujo de datos (a partir de ahora, dataflow) que implementa para optimizar la ejecución de la CNN. Este dataflow es el que decide qué datos se envían a través del array de PEs, cuáles acceden a memoria principal, qué vecinos pueden comunicarse, y qué tipo de datos almacena cada PE de manera que este se mantiene estacionario reutilizándolo durante la mayor cantidad de tiempo posible. En general, los aceleradores tradicionales más conocidos se basan en la implementación de alguno (o varios) de los siguientes dataflows:

- *Weight Stationary (WS)*: este dataflow intenta aprovechar al máximo el reuso de los pesos en una CNN. Para ello, los PEs se agrupan en clusters de tamaño $R \times S$ (siendo $R \times S$ las dimensiones de un filtro) y cada uno de ellos almacena estacionario en un registro un peso de cierto filtro. Una vez que cada uno de los PEs ha leído su respectivo peso, éste se reutiliza durante los próximos $H \times W$ (dimensiones del input ifmap) ciclos. En cada ciclo, una activación de entrada es enviada a todos los PEs (*broadcast*) y ésta es multiplicada por su respectivo peso. Finalmente, las sumas intermedias producidas son transmitidas entre los PEs vecinos para generar la suma final de la activación de salida.
- *Output Stationary (OS)*: este dataflow trata de evitar el envío de sumas intermedias a través del array de PEs, acumulándolas tan pronto como sea posible. Para ello, mantiene la suma intermedia de cada activación de salida almacenada en cada PE. Así, cada uno se encargará de procesar una activación de salida. Con este objetivo, en cada ciclo cada peso es enviado a todos los PEs



(a) Arquitectura temporal (b) Arquitectura sistólica

Fig. 2: Ejemplo general de arquitecturas temporal y sistólica utilizadas para ejecutar CNNs.

(broadcast), mientras que las activaciones son recibidas o bien de memoria (si es un PE que se encuentra en la primera columna del array de PEs) o bien de los vecinos de su izquierda o de arriba. Una vez un PE ha calculado una activación de salida, esta es enviada a memoria principal directamente.

- *Non-Local Reuse (NLR)*: Este tipo de arquitecturas se basan en no mantener ningún tipo de dato estacionario cerca de los PEs y aprovechar todo el área del chip para implementar PEs. Todo el flujo de datos (pesos, activaciones de entrada y activaciones de salida) es leído de memoria por los PEs y transmitido entre ellos en forma de pipeline. Este tipo de arquitecturas están perdiendo interés ya que ofrecen una eficiencia energética más baja debido a la significativa cantidad de accesos a memoria que requieren.

El problema de estos dataflows tradicionales es que no pueden adaptarse a la alta variabilidad de formas que existen entre distintas CNNs, e incluso entre distintas capas de una misma CNN, quedando muchos PEs inutilizados. Por ejemplo, una capa con solo 8 activaciones de salida y 16 PEs utilizando el dataflow OS solo utilizará el 50 % de los PEs (ya que cada uno calcula una activación de salida completa), dejando el resto ociosos.

Recientemente, han surgido nuevas propuestas que tratan de reutilizar varios tipos de datos al mismo tiempo y que ofrecen una mejor eficiencia. Entre ellas, destacan mayormente Eyeriss [8] que trata de reutilizar los 3 tipos de datos mediante su dataflow *Row Stationary (RS)* [23], FlexFlow [11], que permite la agrupación de PEs de modo que cada cluster ejecute un dataflow diferente, y MAERI [12], que usando un dataflow similar al RS de Eyeriss optimiza la red de interconexión para aumentar la flexibilidad del chip e impedir que ningún PE quede inutilizado.

Como vemos, esta gran cantidad de aceleradores, junto con el hecho de que ninguno de ellos libera su implementación, hacen su evaluación y análisis muy complicada para los investigadores del área. Para solucionar esto, en la siguiente sección se presenta *CNN-Sim*, el primer simulador capaz de imitar el comportamiento de un acelerador especializado en dar soporte a la ejecución del procedimiento de inferencia de CNNs de forma realista.

III. CNN-Sim

Como hemos mencionado anteriormente, *CNN-Sim* es una versión extendida de *SCALE-Sim*, el cual es un modelo a nivel de ciclo escrito en el lenguaje de programación Python, que permite simular una arquitectura sistólica capaz de ejecutar el procesamiento de una CNN tradicional como la que se explica en la Sección II.

Este simulador permite realizar experimentos basados en los dataflows WS y OS vistos anteriormente. Además, simula el dataflow *Input Stationary* el cual no es usado actualmente por la comunidad científica. Para modelar estos dataflows, *SCALE-Sim* modela un array bidimensional de PEs (como el que se muestra en la Figura 2b) y tres módulos de memoria SRAM on-chip, de las cuales, dos de ellos se utilizan para leer de forma simultánea los operandos de entrada (pesos y activaciones) y el tercero para escribir las activaciones de salida. Estos módulos de SRAM simulan la técnica de *double buffering* con el objetivo de solapar los accesos a memoria con cómputo en los PEs, evitando así ciclos de parada en el array de PEs.

La configuración del tamaño de los PEs, el dataflow a utilizar y el tamaño de los módulos SRAM vienen dados por un fichero de configuración que puede ser editado por el usuario. La CNN a ejecutar es leída por el simulador a través de otro fichero que contiene por cada capa sus parámetros principales (canales de entrada, número de filtros, tamaño de filtros y de ifmap, etc). Una vez *SCALE-Sim* es puesto en

ejecución, éste lee el fichero de configuración hardware y el fichero con la topología de la red y modela la ejecución del procesamiento de la CNN usando el dataflow especificado por el usuario en el archivo de configuración hardware. El simulador calcula de forma matemática utilizando los parámetros de la red y del hardware, el número de ciclos que tarda la ejecución, trazas con las direcciones de memoria generadas en cada ciclo y el porcentaje de utilización de los PEs. De este modo, *SCALE-Sim* simplemente modela la ejecución pero no la lleva a cabo realmente. Los PEs suponen una memoria SRAM perfecta y no pueden modelar un posible aprovechamiento de optimizaciones bien conocidas basadas en ciertos valores como los ceros, para mejorar la eficiencia energética del acelerador.

Para solucionar estas limitaciones para la exploración del espacio de diseño de aceleradores de CNNs, en este trabajo presentamos una extensión de *SCALE-Sim* denominada *CNN-Sim*. Este nuevo simulador, cuya descripción es mostrada en la Figura 3, permite la simulación de una ejecución real del procedimiento de inferencia de una CNN sobre un acelerador hardware.

Nuestro simulador está compuesto de tres componentes principales: un módulo de entrada, un módulo de ejecución, y un módulo de salida.

- *Módulo de entrada*: Este módulo sirve para poder especificar los datos de entrada que son necesarios para la configuración de las simulaciones. Como se puede ver en la parte izquierda de la Figura 3, se ha integrado una modificación del bien conocido framework de deep learning Caffe [24], permitiéndole al usuario especificar en el fichero de configuración del modelo de CNN (fichero *model.prototxt*) la ruta donde se encuentra el simulador y el fichero de configuración hardware *architecture.cfg* idéntico al utilizado en *SCALE-Sim*. Cuando el usuario ejecute el framework con dicho modelo de CNN, éste lanzará de forma transparente el simulador *CNN-Sim*, cargando los ifmaps de entrada y los pesos en los correspondientes módulos de SRAM simulados. Tras esto, el módulo de ejecución toma el control y ejecuta capa a capa el modelo de CNN especificado.
- *Módulo de ejecución*: Como se explicaba anteriormente, *SCALE-Sim* simplemente calculaba de forma matemática el número de ciclos requeridos para la ejecución del modelo CNN en un determinado dataflow, utilizando los parámetros de configuración hardware del fichero *architecture.cfg*. En lugar de esto, en *CNN-Sim* la arquitectura es realmente simulada. Así, ciclo a ciclo, se realizan las lecturas de SRAM de las activaciones de entrada y de los pesos correspondientes según el dataflow a utilizar, e imita su comportamiento tal y como lo haría un acelerador real, almacenando los valores de salida en su correspondiente módulo de SRAM. Durante la simulación, la herramienta monitoriza y almacena todos los valores leídos y escritos para posteriormente mostrarlos mediante el módulo de salida. Una vez la simulación de todas las capas ha finalizado, el simulador devuelve el control a Caffe, pasándole a éste las activaciones de salida para que pueda proseguir con normalidad su ejecución.
- *Módulo de salida*: Este módulo está preparado para mostrar la salida del simulador cogiendo como entrada los datos que se recopilaron a través del módulo de ejecución. Como puede verse en la parte derecha de la Figura 3, *CNN-Sim*, además de generar el número de ciclos que ha tardado la simulación de cada capa, el porcentaje de utilización de PEs y las trazas de los accesos a las memorias SRAM y DRAM, también genera ficheros de trazas mostrando los valores que ciclo a ciclo han sido leídos y/o escritos por el simulador. De esta forma, el usuario puede visualizarlos y utilizarlos para plantear nuevas técnicas de optimización hardware basadas en los valores que fluyen por el array de PEs.

Para demostrar el potencial de uso de *CNN-Sim*, en la siguiente sección veremos cómo hemos utilizado estos ficheros de salida, con el propósito de analizar la cantidad de ceros que fluyen por los enlaces de la red de interconexión de un acelerador al ejecutar una capa convolucional utilizando los dataflows WS y OS.

IV. Caso de estudio: Influencia del dataflow en el tráfico de red inservible

La existencia de las operaciones inservibles (multiplicaciones por cero) en la ejecución del procedimiento de inferencia de una CNN es ampliamente demostrado por el estado del arte [8]. Sin embargo, todavía no se había llevado a cabo ningún estudio exhaustivo que mostrase la influencia de estos ceros fluyendo por la red de interconexión dependiendo del dataflow utilizado. La razón de esto es que, previamente a *CNN-Sim*, la comunidad científica necesitaba un simulador a nivel de ciclo capaz de analizar los valores computados según el dataflow utilizado.

TABLA II: Parámetros hardware simulados con *CNN-Sim* en nuestro caso de estudio.

Parámetro	valor
Array height	32
Array width	32
ifmap SRAM size	1 MB
filter SRAM size	4 MB
ofmap SRAM size	1 MB
dataflow	OS/WS

Para mostrar el potencial de nuestra herramienta, hemos llevado a cabo un caso de estudio en el cual simulamos la capa convolucional con ID 2 (ver Tabla I) utilizando una única imagen como entrada. Esta capa posee un ifmap de tamaño $256 \times 27 \times 27$ y 384 filtros de tamaño $256 \times 3 \times 3$ cada uno. Los paráme-

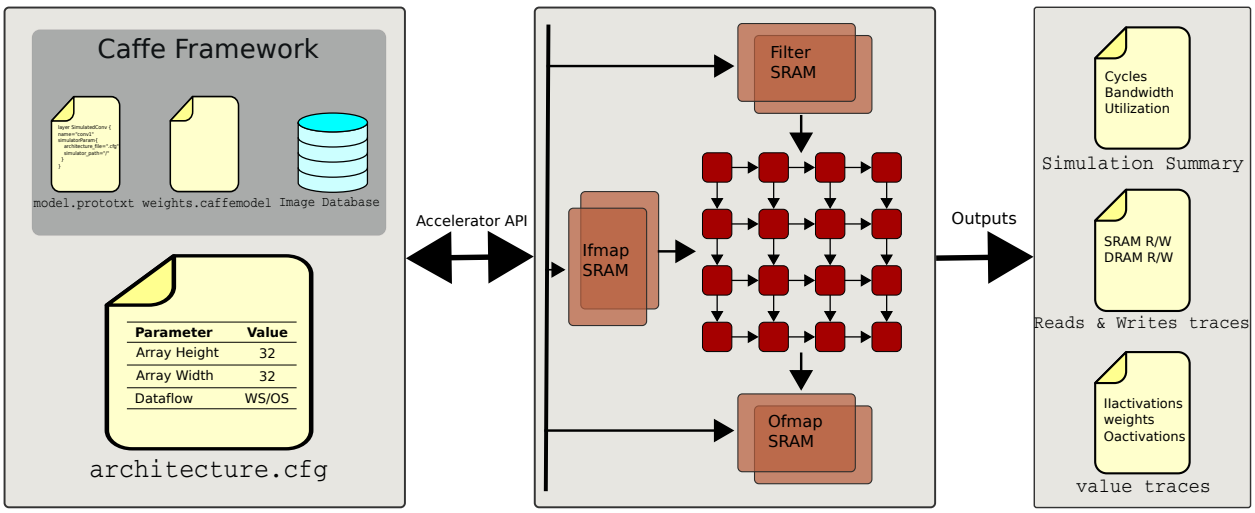


Fig. 3: Descripción de alto nivel de CNN-Sim.

tros hardware que hemos simulado se muestran en la Tabla II.

La Figura 4 muestra el porcentaje de ceros (eje Y) que son leídos y escritos a través de la red de PEs para los dataflows WS y OS por cada intervalo de 20.000 ciclos ejecutados (eje X). En otras palabras, la figura presenta el porcentaje de ceros que reciben todos los PEs desde la red de interconexión teniendo en cuenta tanto los que provienen de memoria, como los que provienen desde PEs vecinos. Como puede apreciarse, el porcentaje de ceros que fluyen por la red es muy significativo para ambos dataflows ya que estos constituyen entre un 31 % y un 34 % del tráfico total. También es importante notar que la cantidad de ceros tiende a ser superior en el dataflow WS. Esto es debido a que éste (como se explicó en la Sección II) mantiene los pesos estacionarios en los PEs, reduciendo el tráfico de red asociado a los mismos (en el dataflow OS estos pesos son enviados cada ciclo a todos los PEs generando más tráfico de red) mientras mantiene la misma cantidad de tráfico debido a las activaciones (las cuales contienen casi todos los ceros). Esto produce que los ceros de las activaciones sean más significativos en el porcentaje de ceros final.

Además de esto, también se puede apreciar que el dataflow WS requiere una mayor cantidad de ciclos que el OS (en concreto 63.873) para ejecutar la capa. Estos ciclos extra, se deben a que el dataflow WS requiere de una primera fase de lectura de los pesos. Tras esta lectura, estos son mantenidos estacionarios y se procede a la lectura de las activaciones de entrada. En el dataflow OS, la lectura de los pesos y de las activaciones se hace de forma simultánea, provocando una terminación más temprana. A pesar de este resultado, hay que tener en cuenta que en una arquitectura real donde se realicen realmente los accesos a memoria DRAM, el factor principal de influencia en el rendimiento del dataflow será la reutilización de los datos en la memoria SRAM y esto dependerá del modo en el que se ajuste la forma de cada capa convolucional a cada dataflow.

V. Conclusiones

En este trabajo se ha presentado *CNN-Sim*, una extensión de *SCALE-Sim* conectada con el framework de deep learning Caffe, que tiene como objetivo la exploración de técnicas arquitecturales para optimizar el procedimiento de inferencia en aceleradores hardware específicos para CNNs. Además, para mostrar el potencial de la herramienta, se ha presentado un caso de uso que utiliza *CNN-Sim* para demostrar la influencia del tráfico de red inservible en los dataflows WS y OS típicos de un acelerador para CNNs. Concretamente, se ha mostrado que los ceros constituyen entre un 31 % y un 34 % respectivamente de todo el tráfico de red para ambos dataflows, resultado muy significativo e interesante que anima a la aplicación de alguna técnica de optimización hardware que reduzca este flujo de ceros por la red, evitando su procesamiento y la utilización de recursos (por ejemplo, entradas en buffers o registros en los PEs), y reduciendo el tiempo de ejecución así como consumo energético.

Como trabajo futuro, inspirados en el acelerador MAERI, actualmente estamos extendiendo la capacidad de la herramienta *CNN-Sim* para simular cualquier tipo de elemento hardware soportado por un acelerador tradicional. Nuestro objetivo final, es proporcionar a los usuarios un framework que pueda ser empleado para configurar y simular cualquier arquitectura específica para CNN actual. De esta forma, será posible la evaluación directa de todas las arquitecturas existentes actualmente, así como la exploración de otros diseños que puedan superar a los anteriores.

Agradecimientos

Este trabajo ha sido financiado por el Ministerio de Ciencia, Innovación y Universidades (MCIU) y la Agencia Estatal de Investigación (AEI), así como con fondos FEDER de la Comisión Europea, mediante los proyectos “RTI2018-098156-B-C53” y “TIN2016-78799-P”. Francisco Muñoz-Martínez es respaldado

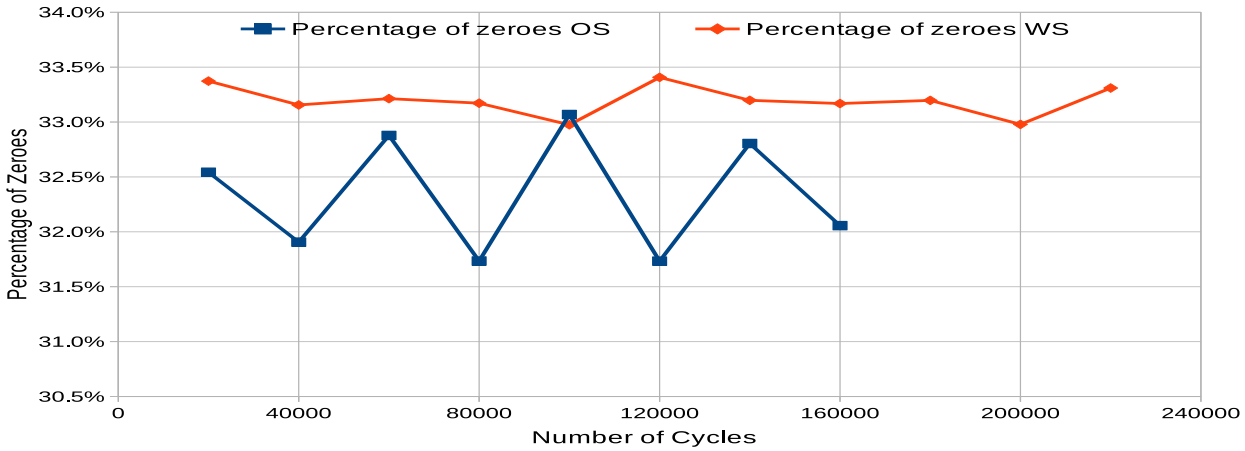


Fig. 4: Porcentaje de ceros que fluyen a través de la malla 2D que interconecta los PEs cuando la tercera capa de la CNN AlexNet (ID 2 en la Tabla I) es ejecutada en *CNN-SIM* usando los dataflows WS y OS.

mediante un contrato predoctoral de formación del personal investigador con referencia 20749/FPI/18 financiado por la Consejería de Empleo, Universidades, Empresa y Medio Ambiente de la CARM, a través de la Fundación Séneca-Agencia de Ciencia y Tecnología de la Región de Murcia.

Referencias

[1] Yann Lecun, Yoshua Bengio, and Geoffrey Hinton, "Deep learning," *Nature*, vol. 521, no. 2, pp. 436–444, May 2015.

[2] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton, "ImageNet classification with deep convolutional neural networks," *International Conf. on Neural Information Processing Systems (NIPS)*, pp. 1106–1114, Dec. 2012.

[3] Li Deng et al., "Recent advances in deep learning for speech research at microsoft," *2013 IEEE International Conf. on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 8604–8608, Dec. 2013.

[4] Chenyi Chen, Ari Seff, Alain Kornhauser, and Jianxiong Xiao, "DeepDriving: Learning affordance for direct perception in autonomous driving," *2015 IEEE International Conf. on Computer Vision (ICCV)*, pp. 2722–2730, Dec. 2015.

[5] Andre Esteva et al., "Dermatologist-level classification of skin cancer with deep neural networks," *Nature*, vol. 542, pp. 115–118, Jan. 2017.

[6] David Silver et al., "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, pp. 484–489, Jan. 2016.

[7] Alex Krizhevsky, Llya Sutskever, and Geoffrey E. Hinton, "ImageNet classification with deep convolutional neural networks," *International Conference on Neural Information Processing Systems (NIPS)*, vol. 1, no. 1, pp. 1097–1105, Dec. 2012.

[8] Yu-Hsin Chen, Joel S. Emer, Tushar Krishna, and Vivienne Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE Journal of Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, Jan. 2017.

[9] Tao Luo, Shaoli Liu, Ling Li, Yuqing Wang, Shijin Zhang, Tianshi Chen, Zhiwei Xu, Olivier Temam, and Yunji Chen, "DaDianNao: A neural network supercomputer," *IEEE Transactions on Computers*, vol. 66, no. 1, pp. 73–88, Jan. 2017.

[10] Srimat Chakradhar, Murugan Sankaradas, Venkata Jakkula, and Srihari Cadambi, "A dynamically configurable coprocessor for convolutional neural networks," *2010 International Symposium on Computer Architecture (ISCA)*, pp. 247–257, June 2010.

[11] Wenyan Lu, Guihai Yan, Jiajun Li, Shijun Gong, Yinhe Han, and Xiaowei Li, "FlexFlow: A flexible dataflow accelerator architecture for convolutional neural networks," *IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 553–564, May 2017.

[12] Hyoukjun Kwon, Ananda Samajdar, and Tushar Krishna, "MAERI: Enabling flexible dataflow mapping over dnn accelerators via reconfigurable interconnects," *International Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pp. 461–475, Mar. 2018.

[13] Ananda Samajdar, Yuhao Zhu, Paul Whatmough, Matthew Mattina, and Tushar Krishna, "SCALE-Sim: Systolic cnn accelerator simulator," *arXiv preprint arXiv: 1811.02883v1 (2019)*, Feb. 2019.

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, "Deep residual learning for image recognition," *arXiv preprint arXiv: 1512.03385v1 (2015)*, Dec. 2015.

[15] Chigozie Enyinna Nwankpa, Winifred Ijomah, Anthony Gachagan, and Stephen Marshall, "Activation functions: Comparison of trends in practice and research for deep learning," *arXiv preprint arXiv: 1811.03378 (2018)*, Nov. 2019.

[16] Jingua Tao and Zidong Du et. al., "Benchip: Benchmarking intelligence processors," *arXiv preprint arXiv: 1710.08315v2 (2017)*, Nov. 2017.

[17] Gysel, Philipp, Pimentel, Jon, Motamedi, Mohammad, Ghiasi, and Soheil, "Ristretto: A framework for empirical study of resource-efficient inference in convolutional neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, pp. 5784 – 5789, 2018.

[18] Hiroya Maeda, Yoshihide Sekimoto, Toshikazu Seto, Takehiro Kashiyama, and Hiroshi Omata, "Road damage detection and classification using deep neural networks with smartphone images," *Computer-Aided Civil and Infrastructure Engineering*, vol. 33, no. 12, pp. 1127–1141, 2018.

[19] Songtao Liu, Di Huang, and Yunhong Wang, "Receptive field block net for accurate and fast object detection," in *European Conference on Computer Vision*. Springer, 2018, pp. 404–419.

[20] Yiting Li, Haisong Huang, Qingsheng Xie, Liguo Yao, and Qipeng Chen, "Research on a surface defect detection algorithm based on mobilenet-ssd," *Applied Sciences*, vol. 8, no. 9, pp. 1678, 2018.

[21] Yuxi Li, Jiuwei Li, Weiyao Lin, and Jianguo Li, "Tiny-dsod: Lightweight object detection for resource-restricted usages," *arXiv preprint arXiv:1807.11013 (2018)*, 2018.

[22] Angshuman Parashar, Minsoo Rhu, Anurag Mukkara, Antonio Puglielli, Rangharajan Venkatesan, Brucek Khailany, Joel Emer, Stephen W. Keckler, and William J. Dally, "SCNN: An accelerator for compressed-sparse convolutional neural networks," *International Symposium on Computer Architecture (ISCA)*, pp. 27–40, June 2017.

[23] Yu-Hsin Chen, Joel S. Emer, and Vivienne Sze, "Using dataflow to optimize energy efficiency of deep neural network accelerators," *IEEE Micro*, vol. 37, no. 3, pp. 12–21, June 2017.

[24] "Caffe website," http://caffe.berkeleyvision.org/.

Evaluación de Rendimiento del Entrenamiento Distribuido de Redes Neuronales Profundas en Plataformas Heterogéneas

Sergio Moreno-Álvarez[1] , Mercedes E. Paoletti[2] , Juan M. Haut[2] , Juan-Antonio Rico-Gallego[1] , Javier Plaza[2] y Juan-Carlos Díaz-Martín[2]

Resumen— ***Asynchronous stochastic gradient descent* es una técnica de optimización comúnmente utilizada en el entrenamiento distribuido de redes neuronales profundas. En distribuciones basadas en particionamiento de datos, se entrena una réplica del modelo en cada unidad de procesamiento de la plataforma, utilizando conjuntos de muestras denominados *mini-batches*. Este es un proceso iterativo en el que al final de cada *mini-batch*, las réplicas combinan los gradientes calculados para actualizar su copia local de los parámetros. Sin embargo, al utilizar asincronismo, las diferencias en el tiempo de entrenamiento por iteración entre réplicas provocan la aparición del *staleness*, esto es, las réplicas progresan a diferente velocidad y en el entrenamiento de cada réplica se utiliza una versión no actualizada de los parámetros. Un alto grado de *staleness* tiene un impacto negativo en la precisión del modelo resultante. Además, las plataformas de computación de alto rendimiento suelen ser heterogéneas, compuestas por CPUs y GPUs de diferentes capacidades, lo que agrava el problema de *staleness*. En este trabajo, se propone aplicar técnicas de equilibrio de carga computacional, bien conocidas en el campo de la Computación de Altas Prestaciones, al entrenamiento distribuido de modelos profundos. A cada réplica se asignará un número de *mini-batches* en proporción a su velocidad relativa. Los resultados experimentales obtenidos en una plataforma heterogénea muestran que, si bien la precisión se mantiene constante, el rendimiento del entrenamiento aumenta considerablemente, o desde otro punto de vista, en el mismo tiempo de entrenamiento, se alcanza una mayor precisión en las estimaciones del modelo. Discutimos las causas de tal incremento en el rendimiento y proponemos los próximos pasos para futuras investigaciones.**

Palabras clave— **Aprendizaje Profundo, Computación de Altas Prestaciones, Entrenamiento Distribuido, Plataformas Heterogéneas, Redes Neuronales**

I. Introducción

El aprendizaje profundo *(Deep Learning)* ha alcanzado niveles de precisión muy altos en áreas como la clasificación de imágenes [1], [2] y el reconocimiento de voz [3], [4]. Estas mejoras son posibles gracias a los avances en las técnicas de entrenamiento, las plataformas HPC y el acceso a grandes conjuntos de datos utilizados para entrenar estos modelos [5].

Los clusters de computación de alto rendimiento permiten acelerar el entrenamiento de estos modelos, que normalmente se basa en técnicas de optimización como Stochastic Gradient Descent (SGD) aplicado a conjuntos de muestras, que se denominan *mini-batches* [6]. Cuando se trata de redes profundas de gran tamaño y grandes conjuntos de muestras, las plataformas paralelas HPC se vuelven indispensables. La paralelización del proceso de entrenamiento y el despliegue en los recursos de la plataforma se realiza mediante dos esquemas principales, conocidos como *model parallelism* (particionamiento del modelo) y *data parallelism* (particionamiento de datos).

El particionamiento del modelo divide el modelo a entrenar, es decir, sus parámetros, entre los recursos computacionales disponibles. Cada proceso, conocido como *worker* o *learner*, entrena una parte del modelo utilizando el mismo *mini-batch* de muestras. Los procesos comunican los resultados intermedios utilizando diferentes estrategias, como por ejemplo, un *pipeline* entre las capas del modelo desplegado en los recursos de la plataforma [7]. Como el entrenamiento es un proceso fundamentalmente secuencial, este tipo de esquemas dificultan el uso eficiente de los recursos computacionales, y su rendimiento estaría limitado por la comunicación entre las diferentes partes del modelo. Este método se utiliza cuando el modelo es lo suficientemente grande como para que no se pueda alojar en memoria de una sola unidad de proceso.

El particionamiento de datos consiste en ejecutar réplicas del modelo completo en cada recurso de computo disponible. Por tanto, cada réplica contiene una copia local de los parámetros a aprender, y se entrena utilizando subconjuntos de datos disjuntos. En cada paso del entrenamiento, los valores resultantes (gradientes) deben comunicarse al resto de las réplicas para combinar los resultados y actualizar los parámetros.

En un entorno distribuido que utiliza particionamiento de datos, el método de optimización denominado *Stochastic Gradient Descent* (SGD) síncrono recorre varias veces el conjunto completo de muestras, en lo que se denominan *épocas*. En cada época, a cada réplica se asigna un subconjunto disjunto de muestras, que a su vez se divide en *mini-batches* para entrenar su propia copia local del modelo. Después de cada *mini-batch*, los gradientes se calculan utilizando una función de pérdida *(loss function)* con respecto a sus valores locales actuales. Finalmente, los procesos se coordinan para combinar sus gradientes locales y actualizar sus parámetros, comenzando

[1]Dpto. de Ingeniería de Sistemas Informáticos y Telemáticos, Universidad de Extremadura, e-mail: smoreno@unex.es y jarico@unex.es.

[2]Dpto. de Tecnología de los Computadores y las Comunicaciones, Universidad of Extremadura, e-mail: mpaoletti@unex.es, juanmariohaut@unex.es, jplaza@unex.es y juancarl@unex.es.

una nueva iteración. La actualización de los valores de los parámetros requiere comunicación entre procesos. Dos métodos comúnmente usados son la utilización de operaciones colectivas de reducción (del tipo `MPI_Allreduce`) [8], o la utilización de un servidor centralizado de parámetros (*parameter server*) [9]. En cualquier caso, la optimización mediante SGD síncrono es determinista con respecto a la actualización de los valores de los parámetros, pues impone puntos de sincronización entre procesos en el momento de combinar los gradientes. Como consecuencia, sin embargo, el tiempo de espera de los procesos en la sincronización tiene un impacto negativo en el rendimiento general.

La optimización ASGD relaja la consistencia de los parámetros al permitir que los procesos combinen gradientes de forma asíncrona. Este esquema desacopla el cómputo y la comunicación, lo que beneficia enormemente el rendimiento del proceso de entrenamiento. Sin embargo, también desacopla los valores de los parámetros en las réplicas. Como consecuencia, el entrenamiento de un *mini-batch* en una réplica puede utilizar una versión desactualizada de los parámetros. La diferencia entre versiones de un parámetro local usado para calcular los gradientes en una réplica y su valor real se conoce como *staleness*. El grado de *staleness* de un parámetro se puede cuantificar asignándole una marca de tiempo en cada cálculo del valor del gradiente. Algunos estudios empíricos ([9]) muestran que un grado de *staleness* bajo no penaliza la precisión del modelo. Mientras que otros trabajos ([10], [11]) proponen mecanismos para reducir el impacto del *staleness* en la precisión del entrenamiento de los modelos, por ejemplo, reforzando negativamente la velocidad de aprendizaje con respecto al valor medio del *staleness* de los parámetros en las réplicas.

Los métodos anteriores de reducción del impacto del *staleness* en la precisión de un modelo están estrechamente relacionados con el comportamiento del método SGD. Sin embargo, hay otro factor que influye en el proceso de aprendizaje de un modelo. Las plataformas HPC suelen ser heterogéneas, y las diferencias en las capacidades computaciones de los recursos asignados a las réplicas tienen impacto en el *staleness* de los parámetros.

En este trabajo estudiamos los efectos en la precisión y en el rendimiento del entrenamiento de redes profundas utilizando un esquema distribuido de particionamiento de datos en plataformas HPC heterogéneas. Nuestro objetivo es estudiar el impacto de la heterogeneidad de la plataforma en la precisión del modelo entrenado, al mismo tiempo que se mejora el rendimiento utilizando mecanismos de equilibrado de carga.

Abordamos el problema en dos pasos. Partiendo de una distribución de réplicas en la plataforma HPC heterogénea, primero, equilibramos la carga de trabajo (datos de entrenamiento) entre las réplicas en proporción a sus capacidades computacionales. Después, usamos el método de optimización ASGD y comunicación basada en operaciones colectivas de los gradientes que proporciona el framework *PyTorch* [12]. Como veremos, este enfoque establece como límite de *staleness* la máxima diferencia relativa entre las velocidades de cómputo de las réplicas. Además, el equilibrado de la carga de trabajo entre las réplicas mejora notablemente el rendimiento del entrenamiento con respecto al SGD síncrono. El efecto general que observamos en nuestros experimentos es que la precisión del modelo aumenta en el mismo número de épocas/*batches* con respecto a la precisión del modelo de entrenamiento en una carga de trabajo no equilibrada, es decir, suponiendo una distribución de carga de trabajo homogénea. Por lo tanto, desde el otro punto de vista, el rendimiento aumenta para alcanzar la misma precisión.

Las principales contribuciones de este trabajo son:

- Evaluar la precisión del entrenamiento de modelos distribuidos utilizando particionamiento de datos cuando las réplicas procesan diferente número de muestras.
- Aplicar técnicas de equilibrio de carga de trabajo (estáticas), comunes en HPC, para distribuir el entrenamiento de modelos profundos, con el objetivo de optimizar el uso de recursos y el tiempo de ejecución.
- Establecer un límite superior para el valor del *staleness* cuando las réplicas se ejecutan en una plataforma heterogénea utilizando una optimización SGD asíncrona.
- Evaluar el impacto de la heterogeneidad computacional de la plataforma en el entrenamiento distribuido de una red neuronal profunda.

El resto de este documento se estructura como sigue. En la Sección II se describe nuestra implementación, incluyendo la distribución de los procesos en el sistema y el procedimiento de entrenamiento del modelo. En la Sección III se detalla la evaluación de nuestro sistema y se presentan los resultados. La Sección IV discute el trabajo relacionado, y finalmente, la Sección V presenta nuestras conclusiones y describe el trabajo futuro.

II. Implementación

Esta sección detalla la implementación de nuestro desarrollo para el entrenamiento de un modelo distribuido con particionamiento de datos, que se evaluará en la sección III.

Desarrollamos un esquema de particionamiento de datos para el entrenamiento distribuido de un modelo en una plataforma HPC heterogénea utilizando el *framework* PyTorch. Asumimos una plataforma heterogénea dedicada, compuesta por un conjunto de nodos de cómputo con diferentes capacidades o velocidades. Actualmente, los nodos de cómputo en clusters heterogéneos comunes combinan diferentes CPUs y GPUs, que se comunican mediante redes de alto rendimiento. Un problema clave es determinar tales capacidades computacionales. Utilizamos la herramienta FuPerMod [13] para determinar de forma

empírica las velocidades de los nodos utilizados para entrenar el modelo.

En el proceso de entrenamiento, cada nodo de cómputo ejecutará una réplica del modelo con una copia local de sus parámetros. Para entrenar el modelo se utiliza un algoritmo de optimización SGD asíncrono. Cada iteración del procedimiento ASGD realiza la siguiente secuencia de tareas:

1. Cada réplica obtiene un conjunto de *batches* aleatorio de tamaño $|B|$ muestras del conjunto total de datos N. Los *batches* asignados entre réplicas en cada época son disjuntos.
2. Las réplicas entrenan el modelo utilizando sus *batches* y calculan sus gradientes basándose en la función de pérdida $l(b|w)$, que calcula el error de la muestra b usando los parámetros en la réplica w con respecto al valor real.
3. Después de calcular los gradientes $\nabla l(b|w)$, cada réplica usa un hilo que realiza una operación de comunicación colectiva para actualizar sus parámetros locales con los valores obtenidos de las otras réplicas.
4. Los parámetros actualizados se calculan utilizando los gradientes y una tasa de aprendizaje (denominada *learning rate*), y comienza una nueva iteración del proceso de entrenamiento.

La Figura 1 muestra el proceso de entrenamiento de un modelo con varias réplicas. A continuación, detallamos los pasos anteriores.

A. Distribución de la Carga de Trabajo

Nuestro enfoque se basa en realizar un equilibrado estático de la carga de trabajo entre los nodos de cómputo involucrados en el entrenamiento, por lo tanto, el primer problema que nos encontramos es determinar la velocidad de dichos nodos. FuPerMod es una herramienta comúnmente utilizada en la optimización del rendimiento de sistemas heterogéos HPC. Se encarga de determinar empíricamente las capacidades computacionales de los nodos involucrados. Para ello, ejecuta un *benchmark* proporcionado por el usuario en cada nodo. Este *benchmark* debe ser representativo de los cálculos realizados en el proceso de entrenamiento, con el fin de obtener mediciones significativas. En nuestro caso, como *benchmark*, utilizamos la función bien conocida GEMM, ejecutada en un rango de diferentes tamaños de problema x. Como salida, FuPerMod devuelve las velocidades de los P nodos (réplicas) en la plataforma, es decir, devuelve un conjunto de P funciones $S = \{s_1(x), s_2(x), \ldots, s_P(x)\}$, que varían a lo largo del rango del tamaño del problema x.

Un aspecto importante es que el conjunto de funciones de velocidad S que caracterizan la plataforma es independiente del proceso de entrenamiento, y se determina estáticamente en un paso previo. Después, las funciones de velocidad S junto con el tamaño específico del conjunto de datos de entrenamiento $|N|$ se utilizan como entradas para la utilidad FuPerMod *partitioner*, que calcula la cantidad de muestras que se deben asignar a cada réplica[1]. Como resultado, se asignan μ_i muestras a cada réplica i, con $\sum_{i=1}^{P} \mu_i = |N|$.

Una vez que se determina el particionamiento de la carga de trabajo, la distribución de dicha carga se realiza al comienzo de cada época. El conjunto de datos (*dataset*) N se divide en P subconjuntos disjuntos según el vector $M = \{\mu_1, \mu_2, \ldots, \mu_P\}$, y se asignan a las réplicas correspondientes. Además, la distribución del conjunto de datos entre réplicas se realiza de tal manera que cada réplica entrena su copia del modelo con todo conjunto de datos a lo largo de las épocas.

B. Entrenamiento y Cálculo de Gradientes

Una vez que el conjunto de datos se divide y se distribuye entre las réplicas en función del vector M, cada réplica entrena iterativamente su copia del modelo en *batches* de tamaño $|B|$. El tamaño de cada *batch* es constante $|B|$, mientras que el número de *batches* utilizados en cada iteración de entrenamiento en una réplica, varía en función de su velocidad relativa.

Nuestra implementación modifica ligeramente el cargador de datos de PyTorch de la siguiente manera. Partimos de la salida de FuPerMod *partitioner*, $M = \{\mu_1, \mu_2, \ldots, \mu_P\}$, siendo μ_i el número de muestras asignadas a la réplica i, y construimos un vector $R = \{r_1, r_2, \ldots, r_P\}$, con $r_i = \lfloor \frac{\mu_i}{min_j \mu_j} \rfloor$, es decir, r_i es el número de *batches* de tamaño $|B|$ que cada réplica podrá entrenar en una iteración en relación con la réplica más lenta.

Para equilibrar la carga de trabajo, las réplicas calculan los gradientes en cada iteración de la siguiente manera:

$$g_i = \frac{1}{r_i \cdot |B|} \sum_{b \in B^\star} \nabla l(b|w_i), \qquad (1)$$

siendo $B^\star$ el número de muestras en r_i *batches* de tamaño $|B|$, y $l(b|w_i)$ la función de pérdida de una muestra b calculada usando los valores de parámetros actuales w_i en la réplica i.

Efectivamente, esto es equivalente a tener tamaños de *batch* desiguales de $r_i \cdot |B|$ en cada réplica ([14]), sin embargo, nuestro método de equilibrado de carga usando diferente cantidad de *batches* se puede implementar como un módulo independiente sin necesidad de modificar el cargador de datos *(dataloader)* de PyTorch. Como principal inconveniente, este método establece una granularidad en el equilibrado de carga dependiente de $|B|$, que, sin embargo, en nuestros experimentos tiene una influencia limitada en el rendimiento.

Finalmente, equilibrar la carga de trabajo modificando el número de *batches* que va a procesar cada réplica en una iteración, garantiza que todas las réplicas terminarán una época al mismo tiempo, con un

[1]Mientras que FuPerMod *partitioner* funciona originalmente con tamaños de problema en bytes, convertimos los datos resultantes en números de muestras, considerando muestras del mismo tamaño.

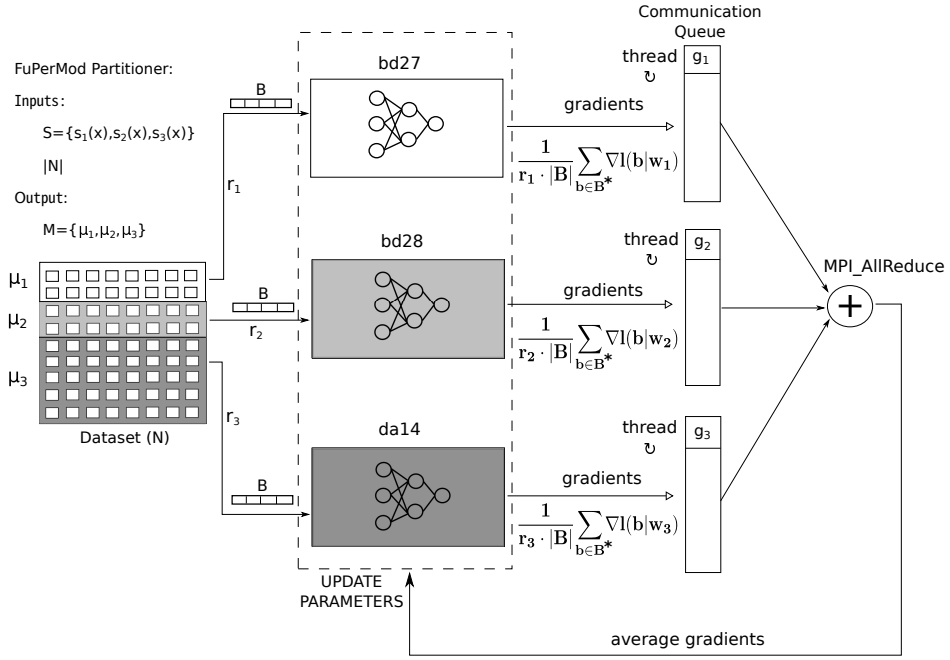


Fig. 1. Estructura de la implementación de un modelo de aprendizaje distribuido utilizando mecanismos de equilibrado de carga y el framework PyTorch. La figura representa una época en el proceso de entrenamiento para $P = 3$ réplicas, con un número desigual de muestras asignadas ($\mu_i, i = 1, \ldots, P$). En cada iteración del entrenamiento, cada réplica utiliza diferente número de *batches* del mismo tamaño ($|B|$) para entrenar su copia del modelo. Para combinar los gradientes resultantes, las réplicas lanzan hilos asíncronos encargados de la comunicación, utilizando la operación colectiva *MPI_Allreduce*.

error que viene determinado por (1) el error en la medición de la velocidad de los nodos con FuPerMod, y (2) la granularidad del equilibrado dependiente de $|B|$, ambos generalmente despreciables.

C. Combinación de Gradientes y Actualización de Parámetros

Existen dos métodos comunes de combinación de gradientes para actualizar la copia de los parámetros del modelo en todas las réplicas. El primero utiliza un servidor de parámetros *(parameter server)* que contiene la copia global actualizada de los parámetros y se encarga de actualizarlos en función de los gradientes recibidos de cada réplica. El servidor de parámetros permite la recepción asíncrona de gradientes. El principal inconveniente de este enfoque es su naturaleza centralizada, que se mitiga manteniendo varios procesos servidores [9]. El segundo enfoque se implementa en el marco PyTorch y consiste en actualizar los parámetros mediante el uso de comunicación colectiva MPI [15]. Cuando una réplica finaliza el cálculo de los gradientes, invoca una operación colectiva de reducción (`MPI _Allreduce`) para combinar los vectores de gradientes de cada réplica y actualizar su copia local de los parámetros. Dicha operación colectiva se encuentra definida en el estándar MPI y es bloqueante, por lo tanto, debido a la necesidad de sincronización tiene un impacto significativo en el rendimiento. La solución asíncrona implementada en PyTorch es utilizar un hilo independiente que se bloquea en la operación colectiva. Antes de la siguiente iteración del entrenamiento, una réplica comprueba si la operación colectiva ha terminado. En caso positivo, actualiza los parámetros locales, y en otro caso realiza una nueva iteración de entrenamiento utilizando la copia actual (por tanto desactualizada) de los parámetros.

En nuestra implementación, una réplica comunica sus gradientes g_i después de procesar $r_i \cdot |B|$ número de muestras, es decir, lanza una operación colectiva después de entrenar r_i *batches*. Este método limita el valor de *staleness*, que dependerá de la heterogeneidad de la plataforma, y tendrá un límite superior de $max_i r_i$. Como conclusión, en el proceso de optimización SGD asíncrono utilizando equilibrado de carga en función de las capacidades computacionales de cada réplica, se reducen los tiempos de espera en la comunicación, mientras se limita el *staleness*, lo que resulta en una mejora general del rendimiento.

III. Experimentación y Evaluación

Esta sección evalúa la implementación propuesta en una pequeña plataforma de prueba. Aunque pequeña, la plataforma sirve como prueba de concepto para obtener resultados iniciales del comportamiento de dicha implementación. Primero, introducimos los elementos de *hardware* y *software* utilizados, y posteriormente discutimos los resultados del entrenamiento distribuido con respecto a la precisión del modelo y el rendimiento del proceso de aprendizaje.

A. Plataforma Experimental

La plataforma heterogénea está compuesta por $P = 3$ nodos multinúcleo conectados por una red Infiniband QDR (4x), en el cluster *Fermi* del centro de cómputo CETA-Ciemat. En este trabajo inicial, elegimos nodos de cómputo tipo CPU con diferente número de núcleos y, por lo tanto, con un rendimiento relativo diferente como se muestra en la Tabla I. Se ejecuta una réplica del modelo en cada nodo de

TABLA I

PLATAFORMA HETEROGÉNEA COMPUESTA DE TRES NODOS (RÉPLICAS) CON DIFERENTES CAPACIDADES COMPUTACIONALES, JUNTO CON LA SALIDA DEL NÚMERO DE MUESTRAS OBTENIDAS CON FUPERMOD PARA UN PROBLEMA TOTAL DE $|N| = 60.000$, Y LAS VELOCIDADES RELATIVAS DE LOS NODOS DE CÓMPUTO PARA EL TAMAÑO DEL CONJUNTO DE DATOS.

Nombre del nodo	MPI Rank	Número de Cores	Tipo de Recurso	Número de Muestras (μ)	Velocidad Relativa
bd27	0	12	CPUs	14.028	1.01
bd28	1	12	CPUs	13.889	1.00
da14	2	24	CPUs	32.083	2.31

cómputo. La columna *número de muestras* contiene el resultado devuelto por FuPerMod *partitioner* para un conjunto de datos de tamaño $|N| = 60.000$ y un vector S que se calcula previamente y que contiene la funciones que determinan la velocidad de las réplicas ejecutando un *benchmark* GEMM para diferentes tamaños de datos.

Los tipos de CPU en los nodos *bd27* y *bd28* son procesadores Intel Westmere de 12 núcleos a 2.53 GHz con 24 GB de RAM, mientras que el nodo nodo *da14* es de tipo Intel Haswell con 24 núcleos a 2.50 GHz y 64 GB de RAM. Los nodos *bd* muestran una ligera diferencia en su velocidad. Esto se puede deber a dos motivos, errores de medición de al determinar la velocidad $s_i(x)$ (que consideramos despreciables) y pequeñas diferencias en el rendimiento del *hardware* y *software* que se ejecuta en los nodos.

En trabajos futuros, planeamos utilizar GPUs, y también una combinación de ambos tipos de procesadores, para explotar la capacidad de cómputo total de la plataforma en el entrenamiento.

B. Descripción del Conjunto de Datos

Utilizamos MNIST [16] como el conjunto de datos para probar nuestra implementación. Este conjunto de datos está compuesto por imágenes en escala de grises que representan dígitos de 0 a 9 escritos a mano. El tamaño del conjunto de entrenamiento es de 60.000 muestras, e incluye un conjunto de verificación de 10.000 muestras, utilizado para calcular la precisión del modelo. Los dígitos se han normalizado y centrado en una imagen de tamaño fijo de $28 \times 28 \times 1$, en la que cada pixel se representa mediante un float de 32 bits. En la Figura 2 se pueden observar algunos ejemplos de las imágenes que componen el conjunto de datos.

C. Resultados

Las réplicas ejecutan el proceso de entrenamiento en su propia copia local del modelo. En particular, el modelo es una red neuronal convolucional (CNN) [17] compuesta de dos partes principales: un extractor de características y un clasificador. Los detalles de la red se muestran en la Tabla II.

Con respecto a la primera parte, se compone de dos etapas de capas convolucionales y de agrupación *(pooling)*, mientras que la segunda parte se compone

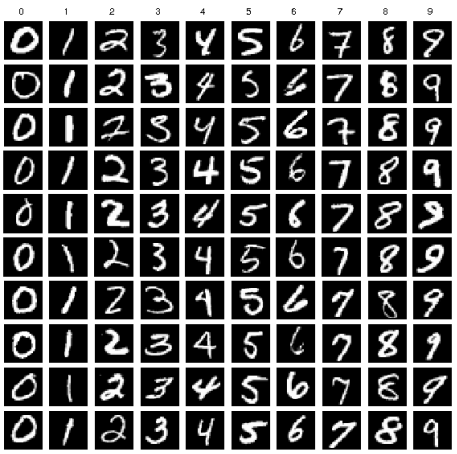


Fig. 2. Muestra de algunas imágenes de digitos escritos a mano en el conjunto MNIST.

TABLA II

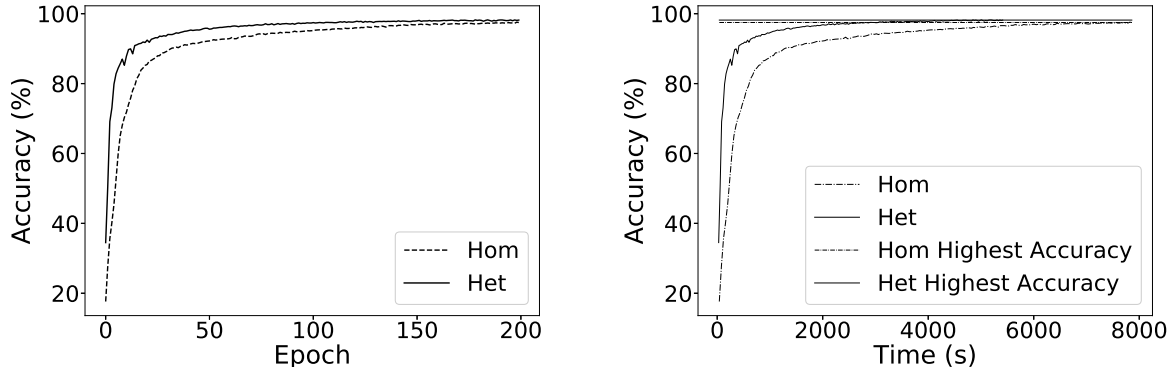
CAPAS DE LA RED NEURONAL CONVOLUCIONAL PARA LA CLASIFICACIÓN DE IMAGENES EN EL CONJUNTO DE DATOS MNIST

ID Capa	*Kernel*/Neuronas	Función Act.	Pooling
Conv1	$20 \times 5 \times 5 \times 1$	ReLU	2×2
Conv2	$50 \times 5 \times 5 \times 20$	ReLU	2×2
FC1	500	ReLU	-
FC2	10	Softmax	-

de dos capas completamente conectadas (FC). Las capas convolucionales bidimensionales emplean como función de activación la Unidad Lineal Rectificada (ReLU, Rectified Linear Unit). Los mapas de características obtenidos en la parte convolucional se transforman en una representación vectorial para alimentar las capas de clasificación.

Definimos la *precisión* del modelo como porcentaje de acierto en la clasificación de las muestras, teniendo en cuenta que la distribución de las clases (digitos) en el conjunto de muestras no contiene sesgo. Por otro lado, medimos el *rendimiento* como el tiempo empleado en el proceso de entrenamiento. Los resultados globales se obtienen tomando la máxima precisión de los modelos y el tiempo máximo de entrenamiento de las réplicas. Comparamos la precisión y el rendimiento del proceso de entrenamiento en la plataforma heterogénea descrita en la sección III-A siguiendo las distribuciones de equilibrado de carga de trabajo tanto homogéneas (todas las réplicas reciben la misma carga) como heterogéneas (cada réplica recibe un número de muestras proporcional a sus capacidades).

La Figura 3 muestra la comparación en la precisión del modelo considerando una distribución homogénea de muestras (*Hom*) entre réplicas, y con equilibrado de carga de acuerdo con sus velocidades (*Het*). El tamaño del *batch* B/P se estable arbitrariamente con $B = 6400$. Hemos evaluado otros tamaños obteniendo resultados muy similares. En la distribución homogénea *Hom*, se usa un solo *batch* por iteración para entrenar el modelo ($r_i = 1, \forall_i$), para más tarde lanzar un hilo para combinar gradientes y ac-



(a) Evolución de la precisión por épocas.

(b) Evolución de la precisión en el tiempo.

Fig. 3. Resultados de precisión obtenidos para el proceso de entrenamiento. *Hom* representa los resultados en la precisión del modelo con un número de *batches* homogéneo. Por el contrario, *Het* representa los resultados en la precisión cuando las réplicas en cada iteración utilizan un número de *batches* proporcional a sus velocidades.

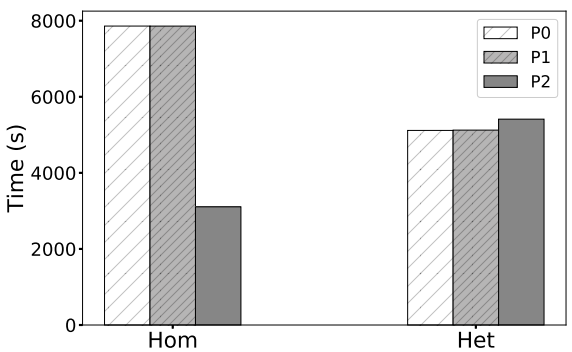


Fig. 4. Tiempo obtenido en 200 épocas de entrenamiento del modelo en las réplicas para ambos métodos de particionamiento de carga: homogéneo y heterogéneo.

tualizar parámetros. En la distribución heterogénea *Het*, el número de *batches* utilizados para entrenar cada réplica cambia, y es proporcional a las velocidades relativas obtenidas utilizando la herramienta *FuPerMod*. Los valores resultantes por iteración son $R = \{1, 1, 2\}$, derivados de la Tabla I. La Figura 3(a) muestra los diferentes valores de precisión para *Hom* y *Het* a lo largo de 200 épocas. La precisión de la distribución homogénea alcanza 96.49 %, mientras que la heterogénea alcanza 98.17 %. Aunque ambos valores de precisión tienden a un mismo valor máximo de aproximadamente 98.65 % en un número mayor de épocas, la gráfica muestra cómo el esquema homogéneo converge en mayor número de épocas. Esto se debe a que el grado de *staleness* causado por las réplicas más lentas repercute negativamente en la precisión, debido a que provocan inestabilidad en la convergencia, como se describe en [18]. La Figura 3(b) muestra cómo la distribución heterogénea *Het* converge más rápido a su valor de máxima precisión en las 200 épocas.

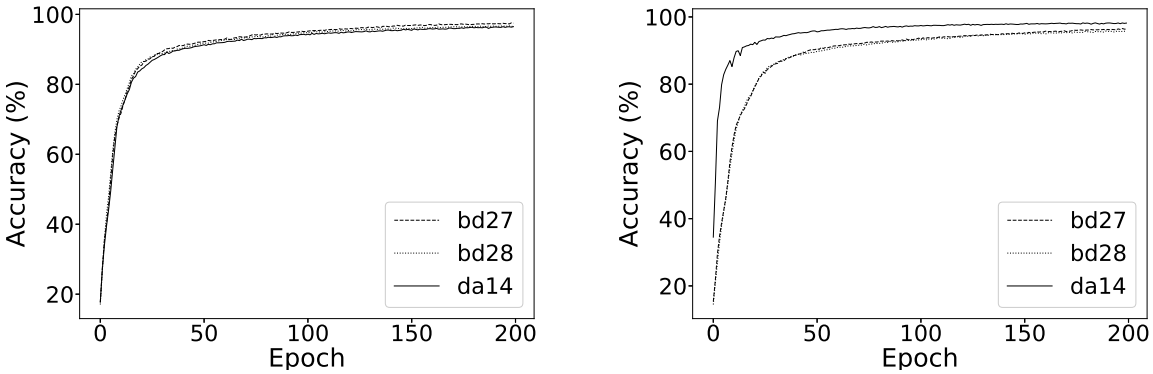
La Figura 4 muestra los detalles de rendimiento de las réplicas después de 200 épocas. En la distribución homogénea nos encontramos con una gran diferencia de tiempo en función de la velocidad de los nodos. Este escenario mejora utilizando un equilibrado de la carga de trabajo en la distribución heterogénea *Het*. Existe una ligera diferencia en los tiempos de *Het* entre nodos, causado por el tamaño de $|B|$, que establece la granularidad en la distribución (ver sección II-B).

La Figura 5 muestra la precisión a lo largo de las épocas para las réplicas involucradas en el entrenamiento de las distribuciones *Hom* y *Het*. Todas las réplicas en la distribución homogénea tienen una precisión similar a lo largo de las épocas, como se muestra en la Figura 5(a), debido a que procesan el mismo tamaño de *batch* con diferentes rendimientos. Sin embargo, en la Figura 5(b) se muestra cómo en una distribución heterogénea, el nodo más rápido alcanza una mayor precisión, porque entrena su modelo con un mayor número de muestras en cada iteración. Por otro lado, las réplicas combinan gradientes con mayor frecuencia y, por lo tanto, tienen un grado de *staleness* más bajo en los parámetros, lo que mejora la precisión y el rendimiento general.

IV. Trabajo Relacionado

El crecimiento en el tamaño de los conjuntos de datos y la cantidad de parámetros en modelos profundos han impulsado el uso de las plataformas HPC para acelerar el entrenamiento. El trabajo [19] proporciona un excelente estudio de las técnicas distribuidas que se utilizan actualmente para paralelizar y distribuir el entrenamiento.

Los principales esquemas de paralelismo, tanto de datos como de modelos, se analizan en el trabajo [20], que propone un entrenamiento distribuido de redes convolucionales utilizando paralelismo de datos en capas convolucionales y paralelismo de modelos en capas completamente conectadas, además de diferentes métodos de sincronización para la actualización de parámetros entre trabajadores. En [9] se propone un algoritmo SGD asíncrono *Downpour* implementado en el *framework DistBelief*. Este algoritmo permite el entrenamiento paralelo de modelos de datos de gran escala utilizando un servidor de parámetros centralizado y *workers* asíncronos. Los autores deter-



(a) Evolución de la precisión en la distribución homogénea. (b) Evolución de la precisión en la distribución heterogénea.

Fig. 5. Resultados obtenidos en términos de precisión de cada uno de las réplicas para las distribuciones homogénea y heterogénea.

minaron que un nivel de tolerancia de *staleness* no afecta significativamente a la precisión del modelo.

Por el contrario, los trabajos [10], [11] proponen un mecanismo para reducir el *staleness* modificando la tasa de aprendizaje utilizando los valores actuales del *staleness* promedio de los gradientes. Proporcionan una discusión sobre la interacción de los hiperparámetros de entrenamiento y las opciones de distribución, utilizando el *framework Rudra*.

El problema del *staleness* también se aborda en [21] con un enfoque diferente. El trabajo propone un entrenamiento basado en un particionamiento de datos entre p réplicas de respaldo, además de las P réplicas principales, utilizando optimización SGD síncrona. Para actualizar los parámetros del modelo, considera las P réplicas más rápidas en el cálculo de los gradientes y descarta el resto. Este enfoque reduce el *staleness* y los tiempos de espera por las réplicas más lentas, sin embargo, el consumo de recursos es mayor.

En cuanto a plataformas heterogéneas, el trabajo [18] estudia la degradación del rendimiento de la optimización SGD en sistemas heterogéneos con respecto a los esquemas de entrenamiento distribuidos homogéneos. Se centra en los sistemas *Stale-Synchronous Parallel*, en los que el protocolo de actualización de parámetros y el servidor de parámetros limitan el grado de *staleness* del sistema. Los autores proponen aplicar tasas de aprendizaje tanto constantes como dinámicas para reducir la inestabilidad en la convergencia del modelo, causada por las réplicas atrasadas, mejorando la precisión y el rendimiento.

Con respecto a la modificación del tamaño del *batch*, *AdaBatch* [22] adapta el tamaño del *batch* a lo largo del proceso de entrenamiento junto con la tasa de aprendizaje, de modo que su relación se mantenga constante, mejorando el rendimiento para *batches* grandes y la precisión para los *batches* pequeños. Sin embargo, este enfoque se aplica en plataformas homogéneas, y no tiene como objetivo contrarrestar la heterogeneidad de la plataforma.

El trabajo [14] propone adaptar los tamaños de los *batches* a las velocidades en cada réplica en una plataforma heterogénea para minimizar los tiempos de espera. A diferencia de nuestra propuesta, este trabajo utiliza un esquema de paralelización *Bulk Synchronous Parallel* en el entrenamiento, esto es, optimización SGD síncrona. La fuente de heterogeneidad (simulada en sus experimentos) proviene del uso no dedicado de los recursos en plataformas *cloud*. El uso del método de optimización SGD síncrono evita el *staleness*. La medición de la velocidad de las réplicas, necesaria para calcular el tamaño de sus *batches* asignados, se logra utilizando una *Recurrent Neural Network* por trabajador, entrenada con valores del uso de memoria y CPU en cada iteración. Vale la pena señalar que nuestro trabajo utiliza SGD asíncrono y explora el impacto del *staleness* en la precisión y el rendimiento en clusters heterogéneos, sin embargo, el trabajo [14] tiene ideas adicionales que planeamos incluir en trabajos futuros, como agregación ponderada de los gradientes en función del tamaño del *batch* para evitar sesgos por muestra en las réplicas, y la evaluación de la implementación en un mayor número de recursos computacionales.

V. Conclusiones y Trabajo Futuro

Este trabajo realiza un estudio preliminar en el contexto del entrenamiento distribuido de redes profundas en plataformas heterogéneas. Proponemos un particionamiento de datos estático, previo al proceso de entrenamiento que ejecutan las réplicas en los recursos de cómputo de la plataforma. Este particionamiento de datos utiliza herramientas tradicionalmente usadas en optimización de aplicaciones HPC como FuPerMod, y asigna a cada réplica un número de muestras proporcional a su velocidad, que se determina con anterioridad. Utilizado junto con un algoritmo de optimización Stochastic Gradient Descent asíncrono, el equilibrado de carga establece un límite superior para el grado de *staleness* entre réplicas, a la vez que reduce los tiempos de espera en los puntos de sincronización y comunicación al final de cada iteración en el entrenamiento. El *staleness* determina el grado en el que una réplica mantiene

parámetros obsoletos o no actualizados para entrenar su copia del modelo. Otros trabajos anteriores analizan cómo el *staleness* afecta negativamente a la precisión del modelo resultante. En nuestra propuesta, dicho grado de *staleness* tiene como límite la máxima diferencia en la velocidad relativa de los recursos heterogéneos asignados a cada réplica, y por tanto, depende de la plataforma.

La implementación de la carga y distribución de datos se realiza a través del *framework* PyTorch, y aprovecha sus capacidades internas basadas en comunicación colectiva MPI para la comunicación asíncrona de los gradientes con el fin de actualizar los parámetros.

Los resultados experimentales, en una plataforma dedicada HPC heterogénea, muestran una mejora en el tiempo de entrenamiento del equilibrado de carga con respecto al método homogéneo, en el que a cada réplica se le asigna un número de muestras uniforme. Desde otro punto de vista, los mecanismos de equilibrado de carga heterogéneos logran una mayor precisión en el mismo tiempo que el mecanismo homogéneo.

Nuestro trabajo futuro se centra en la evaluación de la escalabilidad de la implementación propuesta en dos vertientes. La primera relacionado con el *hardware* y la utilización de un mayor número de nodos heterogéneos de cómputo, incluido el uso de GPUs. La segunda vertiente se relaciona con el *software* y la utilización de un conjunto de datos mayor y más complejo, y una red neuronal convolucional más profunda.

Agradecimientos

Este trabajo ha sido apoyado conjuntamente por los siguientes proyectos e instituciones:

- Ministerio de Educación (Resolución de 26 de diciembre de 2014 y 19 de noviembre de 2015 de la Secretaría de Estado de Educación, Formación Profesional y Universidades, mediante la cual se solicita información para la capacitación de profesores universitarios, de los subprogramas de Capacitación y Movilidad incluidos en el Programa Estatal para las Promociones de Talento y su Empleabilidad, en el marco del Plan Estatal de Investigación e Innovación Científica y Técnica de 2013-2016.
- Por el Fondo Europeo de Desarrollo Regional 'Una manera de hacer Europa'(FEDER) y el gobierno local de Extremadura (Ref. IB16118).
- Por la Administración Local de Extremadura (Ref. 297/2014, ayuda para llevar a cabo actividades de investigación y desarrollo tecnológico y para el desarrollo, la difusión y la transferencia de conocimientos para los grupos de investigación de Extremadura, Ref. GR15005).
- Por el Proyecto MINECO TIN2015-63646-C5-5-R a su vez con el Fondo Europeo de Desarrollo Regional 'Una manera de hacer Europa' (FEDER).
- Por las instalaciones informáticas del Centro de Investigación de Tecnologías Avanzadas de Extremadura (CETA-CIEMAT), financiado por el Fondo Europeo de Desarrollo Regional (FEDER).

Referencias

[1] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds., pp. 1097–1105. Curran Associates, Inc., 2012.

[2] Gao Huang, Zhuang Liu, and Kilian Q. Weinberger, "Densely connected convolutional networks," *CoRR*, vol. abs/1608.06993, 2016.

[3] Dong Yu and Li Deng, *Automatic Speech Recognition: A Deep Learning Approach*, Signals and Communication Technology. Springer, London, 2015.

[4] Dario Amodei, Sundaram Ananthanarayanan, Rishita Anubhai, and more., "Deep speech 2: End-to-end speech recognition in english and mandarin," in *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*. 2016, ICML'16, pp. 173–182, JMLR.org.

[5] Geoffrey Fox, Judy Qiu, Shantenu Jha, Saliya Ekanayake, and Supun Kamburugamuve, "Big data, simulations and hpc convergence," in *Big Data Benchmarking*, Tilmann Rabl, Raghunath Nambiar, Chaitanya Baru, Milind Bhandarkar, Meikel Poess, and Saumyadipta Pyne, Eds., Cham, 2016, pp. 3–17, Springer International Publishing.

[6] Quoc V. Le, Jiquan Ngiam, Adam Coates, Abhik Lahiri, Bobby Prochnow, and Andrew Y. Ng, "On optimization methods for deep learning," in *Proceedings of the 28th International Conference on International Conference on Machine Learning*, USA, 2011, ICML'11, pp. 265–272, Omnipress.

[7] Yanping Huang, Yonglong Cheng, Dehao Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V. Le, and Zhongqian Chen, "Gpipe: Efficient training of giant neural networks using pipeline parallelism," *CoRR*, vol. abs/1811.06965, 2018.

[8] Alexander Sergeev and Mike Del Balso, "Horovod: fast and easy distributed deep learning in tensorflow," *CoRR*, vol. abs/1802.05799, 2018.

[9] Jeffrey Dean, Greg S. Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Quoc V. Le, Mark Z. Mao, Marc'Aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, and Andrew Y. Ng, "Large scale distributed deep networks," in *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, USA, 2012, NIPS'12, pp. 1223–1231, Curran Associates Inc.

[10] Suyog Gupta, Wei Zhang, and Fei Wang, "Model accuracy and runtime tradeoff in distributed deep learning: A systematic study," in *Proceedings of the 26th International Joint Conference on Artificial Intelligence*. 2017, IJCAI'17, pp. 4854–4858, AAAI Press.

[11] Wei Zhang, Suyog Gupta, Xiangru Lian, and Ji Liu, "Staleness-aware async-sgd for distributed deep learning," in *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*. 2016, IJCAI'16, pp. 2350–2356, AAAI Press.

[12] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer, "Automatic differentiation in pytorch," in *NIPS-W*, 2017.

[13] David Clarke, Ziming Zhong, Vladimir Rychkov, and Alexey Lastovetsky, "Fupermod: A framework for optimal data partitioning for parallel scientific applications on dedicated heterogeneous hpc platforms," in *Parallel Computing Technologies*, Victor Malyshkin, Ed., Berlin, Heidelberg, 2013, pp. 182–196, Springer Berlin Heidelberg.

[14] Chen Chen, Qizhen Weng, Wei Wang, Baochun Li, and Bo Li, "Fast distributed deep learning via worker-adaptive batch sizing," in *Proceedings of the ACM Symposium on Cloud Computing*, New York, NY, USA, 2018, SoCC '18, pp. 521–521, ACM.

[15] Richard L. Graham, Timothy S. Woodall, and Jeffrey M. Squyres, "Open mpi: A flexible high performance mpi," in *Parallel Processing and Applied Mathematics*, Roman Wyrzykowski, Jack Dongarra, Norbert Meyer, and Jerzy

Waśniewski, Eds., Berlin, Heidelberg, 2006, pp. 228–239, Springer Berlin Heidelberg.

[16] Yann LeCun, Corinna Cortes, and Christopher JC Burges, "The mnist database of handwritten digits, 1998," *URL http://yann. lecun. com/exdb/mnist*, vol. 10, pp. 34, 1998.

[17] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[18] Jiawei Jiang, Bin Cui, Ce Zhang, and Lele Yu, "Heterogeneity-aware distributed parameter servers," in *Proceedings of the 2017 ACM International Conference on Management of Data*, New York, NY, USA, 2017, SIGMOD '17, pp. 463–478, ACM.

[19] Tal Ben-Nun and Torsten Hoefler, "Demystifying parallel and distributed deep learning: An in-depth concurrency analysis," *CoRR*, vol. abs/1802.09941, 2018.

[20] Alex Krizhevsky, "One weird trick for parallelizing convolutional neural networks," *CoRR*, vol. abs/1404.5997, 2014.

[21] Jianmin Chen, Rajat Monga, Samy Bengio, and Rafal Jozefowicz, "Revisiting distributed synchronous sgd," in *International Conference on Learning Representations Workshop Track*, 2016.

[22] Aditya Devarakonda, Maxim Naumov, and Michael Garland, "Adabatch: Adaptive batch sizes for training deep neural networks," *CoRR*, vol. abs/1712.02029, 2017.

Análisis de rendimiento y eficiencia energética de arquitecturas ARM de bajo consumo

Pavel Nichita, Sergio Afonso, Alberto Cabrera, Francisco Almeida, Vicente Blanco y Dagoberto Castellanos-Nieves [1]

Resumen— **En la actualidad, la optimización de rendimiento de aplicaciones tanto en entornos de altas prestaciones como en dispositivos móviles se ha centrado cada vez más en la reducción del consumo energético. Ha habido un aumento exponencial de arquitecturas de bajo consumo, entre las que destaca la presencia de ARM, gracias a los dispositivos móviles y ahora estas arquitecturas se están adoptando en entornos HPC. Incorporar estos nuevos sistemas enfocados al uso diario a un entorno de computación de altas prestaciones requiere un análisis de sus características. Hemos desarrollado un entorno de análisis de rendimiento y consumo energético que facilita el estudio de los dispositivos disponibles con el que se pueden obtener fácilmente una serie de métricas de tiempo y energía sincronizadas con eventos especificados en el software a analizar. Se ha utilizado nuestro entorno para realizar un análisis de las características de varios *kernels* de los *NAS Parallel benchmarks* y hemos obtenido resultados que indican que la estrategia óptima, desde el punto de vista de la eficiencia energética, para ejecutar aplicaciones en este tipo de arquitecturas consiste en paralelizar, incluso en casos donde un mayor tiempo de ejecución implica un menor consumo energético.**

Palabras clave— **Análisis energético; Arquitectura bajo consumo; System–on–chip; Computación de Altas Prestaciones**

I. Introducción

En los últimos años la arquitectura ARM ha evolucionado mucho en rendimiento y consumo energético. Actualmente, estas arquitecturas de bajo consumo se encuentran presentes tanto en dispositivos de uso cotidiano como en centros de computación de altas prestaciones. Algunos servidores de estas arquitecturas incluso cuentan con hasta 64 núcleos [1], que permiten realizar desde computación comercial en la nube hasta computación científica de alto rendimiento.

En entornos de desarrollo se suelen utilizar pequeñas placas de bajo consumo con arquitecturas similares a las encontradas en dispositivos móviles. Un ejemplo de estos sistemas son las *Single Board Computer* (SBC), que consumen pocos recursos energéticos y tienen bajo coste. Las SBC, son sistemas completos cuyo procesador se encuentra integrado en un *system–on–chip* (SoC), que integra procesador, acelerador gráfico y en algunos casos, procesadores específicos adicionales para audio o vídeo entre otros.

Los SBC son una forma rápida y eficiente de evaluar rendimiento y consumo energético en arquitecturas SoC, que en nuestro caso son todas basadas en procesadores ARM. Además permiten evaluar varios entornos con sistemas operativos diferentes, tanto Linux como Android. Poder ejecutar un algoritmo, en un lenguaje determinado y un entorno variable, permite evaluar su desempeño sin requerir mucho esfuerzo de adaptación para portar el código experimental.

El presente artículo esta organizado de la siguiente forma: la sección II aborda el trabajo relacionado en análisis de rendimiento y energía del estado del arte. En la sección III describimos nuestro sistema de obtención de medidas de rendimiento y consumo energético. En la sección IV describimos los algoritmos utilizados para el análisis y las arquitecturas incluidas. En la sección V mostramos las métricas y resultados extraídos de la experimentación. Concluimos en la sección VI donde resumimos los hitos alcanzados y planteamos el trabajo futuro.

II. Estado del arte

Tradicionalmente, la eficiencia energética en computación de altas prestaciones se ha mejorado minimizando el tiempo de ejecución de las aplicaciones paralelas. Múltiples librerías implementan paquetes de álgebra lineal y hacen uso de distintos modelos computacionales para lograr este objetivo. *Plasma* [2] y *MAGMA* [3] son ejemplos de librerías que ofrecen para arquitecturas de memoria compartida y arquitecturas heterogéneas. Este esfuerzo para optimizar los recursos computacionales desde el punto de vista energético también se ha propagado a los sistemas embebidos y arquitecturas móviles, como es el caso de *MAGMA Embedded* [4].

Por otro lado, existe una iniciativa en computación de altas prestaciones que trata de impulsar el interés por la minimización del consumo energético. Esto se debe a que alternativas puramente basadas en la maximización del rendimiento para la mejora de la eficiencia energética son insuficientes para explotar eficientemente los sistemas exaescalares del futuro [5]. Esta creciente tendencia queda plasmada en el desarrollo de numerosas técnicas para reducir la potencia necesaria para ejecutar aplicaciones. HEROS [6] presenta un algoritmo de reparto de trabajo para asignar eficientemente los recursos en sistemas heterogéneos. Mediante un ajuste dinámico de la frecuencia y el voltaje (DVFS) de los procesadores, se han desarrollado algoritmos de planificación para sistemas de cloud, minimizando el impacto en el rendimiento del servicio ofrecido [7], [8].

En el ámbito del desarrollo de aplicaciones para dispositivos móviles, la utilización de librerías de álgebra lineal para la resolución de problemas de ámbito científico y simulaciones es mucho menos fre-

[1]HPC Group. ETS de Ingeniería Informática. Universidad de La Laguna, ULL. La Laguna. 38270 Tenerife. Spain, e-mail: pnichita@ull.edu.es.

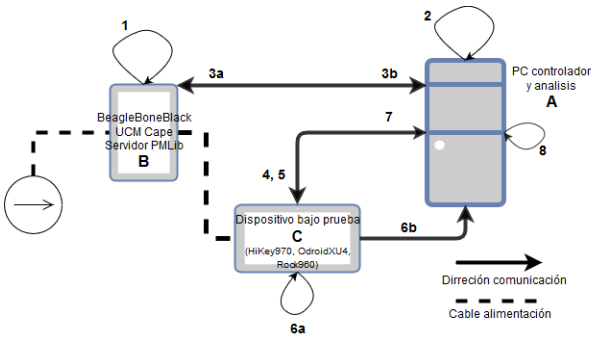


Fig. 1. Sistema de medidas

cuente. Sin embargo, otras aplicaciones de alto rendimiento como la generación de gráficos [9] y la visión por computador [10], el procesamiento de imágenes o las inferencias a través de redes neuronales artificiales [11] son de gran relevancia. Este tipo de aplicaciones se ven significativamente beneficiadas, desde el punto de vista de la eficiencia energética y el rendimiento, por el uso adecuado de los diferentes procesadores que podemos encontrar en el SoC de cualquier dispositivo móvil moderno [12]. Es en estos casos donde la importancia de librerías de alto rendimiento en dispositivos móviles se ve reflejada.

Para extender este tipo de librerías y técnicas a otras arquitecturas de bajo consumo, es necesario analizar la capacidad de los sistemas emergentes mediante análisis de viabilidad. Este análisis debe ser tanto del hardware como de las aplicaciones, y desde el punto de vista tanto energético como de rendimiento. Se ha demostrado que las arquitecturas ARM son viables en rendimiento para la ejecución de numerosas aplicaciones [13]. También se han realizado análisis de arquitecturas híbridas, compuestas de CPU y GPU, presentes en los sistemas más potentes de la lista *Top500* [14]. Nuestro trabajo se centra en otro tipo de arquitecturas de bajo consumo basadas en *system–on–chip*, que pueden ser utilizadas como una alternativa a las arquitecturas más utilizadas, que podrían implementar las técnicas presentes en la literatura.

III. Obtención de métricas

El sistema de obtención de métricas, ilustrado en la figura 1, se compone de un sistema de gestión de la experimentación (**A**), un dispositivo de medida basado en Beagle Bone Black con un servicio de la librería *PMlib* [15], [16] (**B**) y el dispositivo a estudiar en el que se realizarán las pruebas (**C**).

La alimentación de todos los dispositivos estudiados se pasa a través de la Beagle Bone Black, que cuenta con un *CAPE* para obtener las métricas de consumo. Los datos son recolectados con una aplicación cliente servidor que hemos desarrollado en python para el entorno de medida. A través de la API de *PMlib* se recolectan los datos de consumo de cada prueba. En nuestro servidor, hacemos uso de *flask* para identificar y sincronizar las diferentes secciones deseadas del código a instrumentar.

Se utiliza un sistema de colas para lanzar la recolección de datos y las pruebas a analizar. Para poder ejecutar simultáneamente y sin solapamiento en varios dispositivos hay una cola asociada a cada uno de ellos. En el sistema a analizar se minimiza en la medida de los posible la ejecución de servicios que puedan afectar a la experimentación **(C)**.

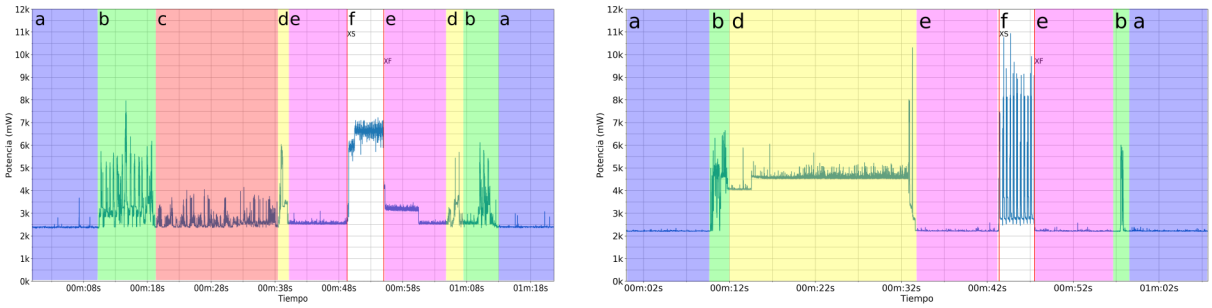
Las pruebas deben ser lo más deterministas posible por lo que evitamos la interacción humana mientras se realizan. En Android esto se hace posible con el uso de **Appium** [17], un framework cliente-servidor para automatizar y probar interfaces gráficas en dispositivos móviles. En el caso de Linux, el control se hace mediante scripts en **bash** y utilizando **ssh** para enviar los comandos de forma remota hacia el cliente que ejecuta la prueba.

En la figura 1, se ilustran los flujos de comandos entre los diferentes elementos que intervienen en la experimentación:

1. **B** tiene ejecutando siempre el servicio de *PMlib*.
2. El dispositivo **A** inicia el servidor de recolección los datos.
3. Mecanismo de obtención de datos:
   a) **A** se conecta a **B** y consulta los datos de consumo.
   b) **B** envía a **A** los datos de consumo.
4. **A** copia y/o instala la aplicación en **C**.
5. **A** indica que se ejecute la prueba en **C**.
6. Ejecución de la prueba:
   a) Si **C** es Android, Appium lanza la aplicación, establece los parámetros de entrada de la prueba y la ejecuta. Si **C** es Linux se lanza la aplicación con los mismo parámetros mediante **ssh/scp**.
   b) Durante la ejecución de la prueba se envían mensajes http a **A** para marcar las zonas de cómputo sobre los datos de **B** .
7. Fin de la prueba, **A** guarda los datos, borra las aplicaciones, cierra el servidor de recolectar y marcar.
8. **A** analiza los datos, los transforma, calcula el uso de energía y genera las correspondientes gráficas.

En las figuras 2(a) y 2(b) se puede ver un ejemplo de las gráficas de consumo energético, donde el eje X es tiempo y el eje Y es la potencia consumida en milivatios. Hemos ilustrado con zonas en diferentes colores cada una de las fases de la ejecución de una prueba:

- Zona **a**: pre y post ejecución, el dispositivo esta en su estado por defecto.
- Zona **b**: proceso de configuración de Appium en caso de Android: en la fase pre–ejecución se instala y lanza la aplicación de pruebas, en la fase post–ejecución se cierra la aplicación. En caso de Linux: en la fase pre–ejecución se copia y extrae el ejecutable, en la fase post–ejecución se borra el ejecutable.
- Zona **c**: en caso de Android es la fase de interacción con la aplicación (entrada de datos): en esta fase es donde se seleccionan los parámetros de



(a) Android, NPB MG (b) Linux, NPB IS

Fig. 2. Ejemplo estructura gráfica

TABLA I
DETALLES DE LOS DISPOSITIVOS

Dispositivo	**ODroidXU4**	**HiKey970**	**Rock960**
Núcleos	4xA15, 4xA7	4xA73, 4xA53	2xA72, 4xA53
Velocidad de reloj	2.0GHz, 1.3GHz	2.36GHz, 1.8GHz	1.8GHz, 1.4GHz
Tamaño palabra	32bits	64bits	64bits
CPU	Exynos5	Kirin 970	RK3399
Fabricante	Samsung	HiSilicon	Rockchip
Memoria	2GB, LPDDR3@933MHz	6GB, LPDDR4X@1866MHz	4GB, LPDD3@1866MHz
Linux ver.	Ubuntu 16.04 kernel 4.14	Debian 9 kernel 4.9	Debian 9 kernel 4.4
Android ver.	Android 4.4.4 kernel 3.10	Android 9 kernel 4.9	Android TV Box 7.1 kernel 4.4

ejecución como la prueba, nº hilos, tamaño de problema, etc. En caso de Linux no existe porque no hay interactividad. Los parámetros del experimento se especifican en la propia línea de comandos.

- Zona **d**: según la prueba utilizada puede haber una fase de preparación y otra de borrado de datos. En la figura 2(a) se ejecuta la prueba NPB MG que tiene una preparación corta comparada con IS (figura 2(b)), que tiene una fase de preparación más extensa.
- Zona **e**: especificamos una pausa de 10 segundos antes y después de la fase de cómputo de la prueba que queremos analizar. Se utiliza para poder identificar de forma visual donde se realiza la computación bajo estudio en las gráficas generadas a posteriori.
- Zona **f**: la fase de cómputo de interés de la prueba. De aquí extraemos los datos para el estudio.

El tamaño de cada zona es variable, depende del dispositivo, de la prueba y sus parámetros. Para una mejor visualización se realizan marcas con unas lineas perpendiculares rojas el principio **XS** y final **XF** del cómputo de la prueba a estudiar.

IV. Experiencia Computacional

Las pruebas se han realizado en varios dispositivos experimentales de 96boards, cuyas especificaciones hemos resumido en la tabla I. Estos dispositivos nos permiten analizar diferentes diseños hardware ARM, que cuentan con diferentes *SoCs*, tamaño de palabra y velocidad de la memoria. Otro aspecto interesante, es la capacidad de cada dispositivo de ejecutar tanto Linux como Android como sistema operativo, aunque el soporte y las versiones no son uniformes. Podemos encontrar versiones desde Android *KitKat* hasta Android *Pie* y kernels Linux desde la versión 3.10 hasta la versión 4.14. A nivel de software, el soporte es limitado en cada sistema Android, por lo tanto se ha utilizado la versión por defecto en cada uno de ellos. En el caso de Linux, se ha instalado JRE de Oracle, versión 8 update 202 para homogeneizar el runtime de Java.

Las pruebas seleccionados para medir el rendimiento de los dispositivos bajo estudio forman parte de la suite *NAS Parallel Benchmarks (NPB)* [18], en su versión 3.0, la única disponible en Java. De entre todos los kernels disponibles, se han seleccionado la versión java de tres casos según su comportamiento computacional.

- El *Block Tri-diagonal solver* (BT) se ha seleccionado por ser un kernel puramente computacional, al tratarse de operaciones de álgebra lineal.
- El *Integer Sort* (IS) es un kernel que realiza accesos a memoria frecuentemente y de forma aleatoria. Realiza una ordenación de múltiples indices enteros.
- Por último, el *Multi-Grid on a sequence of meshes* (MG) es un problema intensivo en memoria, y además implica numerosas comunicaciones entre procesos tanto de corta como larga distancia.

Han sido necesarias modificaciones en NPB para poder marcar los diferentes puntos críticos en la ejecución de estas pruebas. El código responsable de la fase de computo (zona **f** en las figuras 2(a) y 2(b)) sigue en su versión original. Si existiese versión Java de NPB EP (*Embarrassingly Parallel*), se podría realizar una experimentación con numerosos dispositivos para estudiar la escalabilidad de estas arquitecturas.

TABLA II
La mediana de los tiempos de ejecución y consumos de energía

Tiempo de ejecución en segundos

		HiKey970				ODroidXU4				Rock960			
	NPB \ Hilos	1	2	4	8	1	2	4	8	1	2	4	6
Android	BT	147.63	104.74	**77.74**		92.14	94.23	**73.83**		126.65	106.05	**64.59**	
	IS	17.12	**10.80**	11.17		26.69	17.48	**11.88**		19.70	**12.67**	13.69	
	MG	52.09	36.55	**31.43**		176.95	143.49	**94.74**		41.23	37.32	**31.79**	
Linux	BT	61.86	52.96	47.70	**40.80**	311.43	176.69	142.32	**102.02**	90.49	103.34	75.28	**54.85**
	IS	13.28	7.37	**5.27**	5.60	12.96	**8.84**	23.16	14.56	14.92	**12.81**	13.17	13.49
	MG	37.09	22.98	19.63	**19.48**	73.95	**39.80**	83.28	48.41	48.73	31.39	29.58	**28.92**

Energía utilizada en joules

		HiKey970				ODroidXU4				Rock960			
	NPB \ Hilos	1	2	4	8	1	2	4	8	1	2	4	6
Android	BT	997	696	**520**		546	430	**311**		760	607	**405**	
	IS	121	**80**	83		142	113	**93**		119	**79**	90	
	MG	362	257	**217**		1192	1141	**809**		279	238	**220**	
Linux	BT	436	363	322	**283**	1598	1196	1076	**769**	460	458	356	**283**
	IS	99	63	**49**	52	71	**61**	97	82	86	**82**	85	87
	MG	270	171	146	**145**	468	314	400	**283**	274	180	172	**167**

TABLA III
Rangos intercuartílicos de los tiempos de ejecución y consumos de energía

Rango intercuartílico del tiempo de ejecución

		HiKey970				ODroidXU4				Rock960			
	NPB \ Hilos	1	2	4	8	1	2	4	8	1	2	4	6
Android	BT	12.94	5.2	2.62		1.65	4.03	1.77		0.72	1.58	0.87	
	IS	0.41	0.92	0.84		0.3	0.35	0.51		0.29	0.25	0.43	
	MG	2.18	2.86	0.73		2.18	2.68	1.11		1.1	2.86	3.98	
Linux	BT	16.31	8.12	8.99	9.55	2.5	1.4	0.29	0.46	2.31	3.44	3.36	1.98
	IS	0.12	0.09	0.14	0.73	0.17	0.15	0.49	0.5	0.11	0.11	0.13	0.24
	MG	5.45	3.03	4.23	3.38	4.05	2.08	0.23	0.7	0.52	1.67	0.98	0.85

Rango intercuartílico de la energía utilizada

		HiKey970				ODroidXU4				Rock960			
	NPB \ Hilos	1	2	4	8	1	2	4	8	1	2	4	6
Android	BT	73.95	30.2	14.92		7.95	7.01	5.83		7.70	5.04	6.19	
	IS	2.49	5.6	6.4		3.99	2.13	5.36		2.9	1.88	5.49	
	MG	16.65	18.01	3.9		12.3	20.13	23.94		4.36	23.85	24.11	
Linux	BT	65.76	46.37	55.65	52.93	22.9	12.32	7.98	7.68	8.88	21.55	8.94	8.74
	IS	1.91	1.84	1.74	2.88	1.61	1.17	1.37	1.38	0.76	0.93	0.76	1.46
	MG	15.18	13.06	26.31	18.72	22.25	16.39	3.62	1.47	2.11	6.15	3.99	3.38

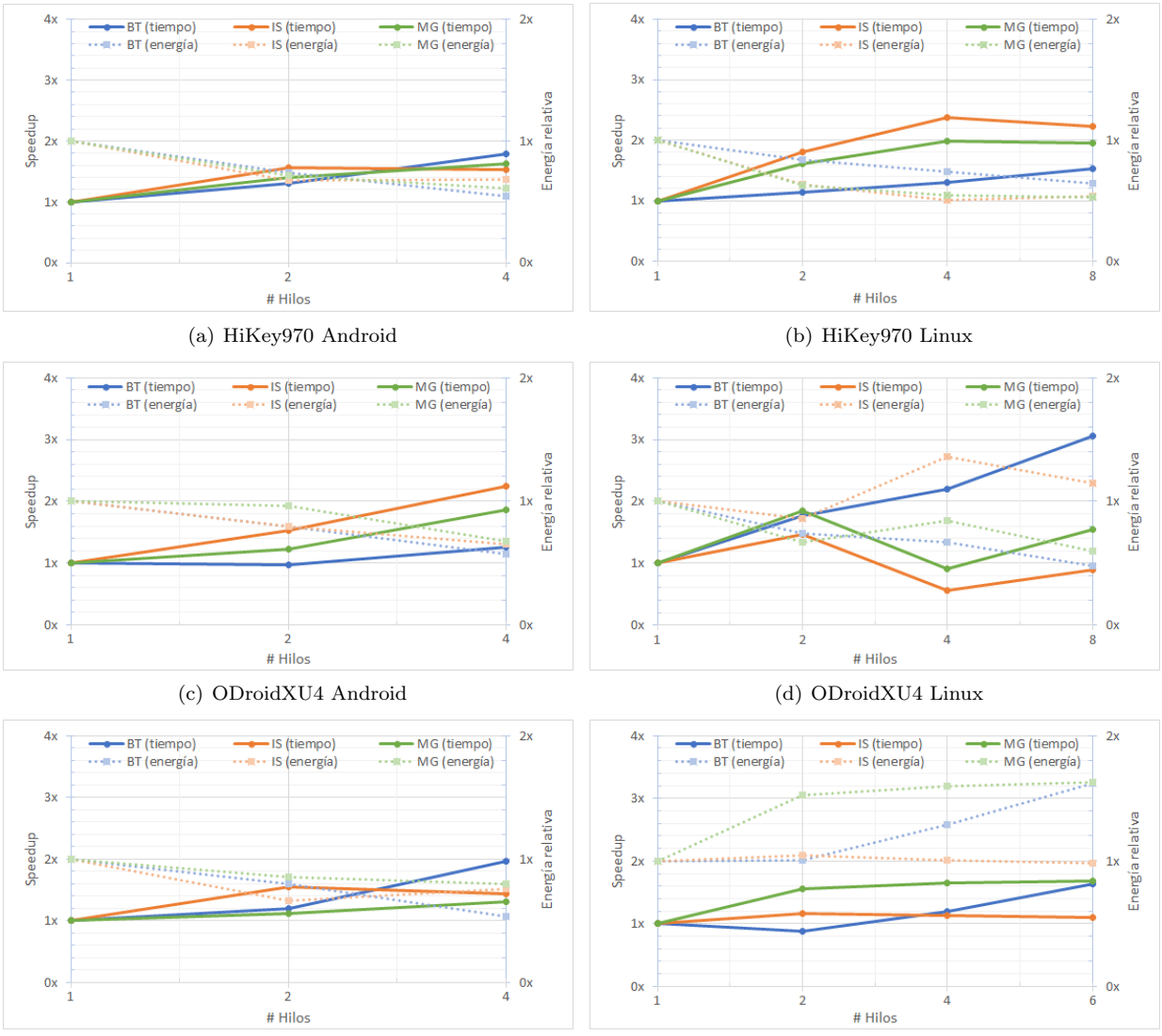
A nivel experimental, no hemos podido homogeneizar el número de hilos para las dos versiones de sistema operativo. Android solo permite la ejecución de hasta un máximo de 4 hilos, aunque la plataforma en la que se ejecuta disponga de más núcleos. No es así el caso de Linux, donde hemos podido aprovechar el número total de elementos computacionales disponibles en estas arquitecturas. Los tamaños de problema, denominados clases en la nomenclatura de NPB, son el tamaño **B** para IS y MG, y para BT el tamaño **W**. Estos tamaños de problema para los kernels elegidos son suficientemente representativos para analizar el rendimiento y la eficiencia energética en nuestro entorno experimental. Además en Android no se puede especificar un tamaño de heap de memoria para el JRE, por lo que debemos ajustarnos a las limitaciones del sistema.

Cada kernel se ha ejecutado un mínimo de 20 veces en cada dispositivo con Android y Linux para eliminar la variabilidad de las métricas. Además nos permite analizar los rangos intercuartílicos para las medidas obtenidas (tabla III).

V. Resultados

La tabla II enumera la mediana de los tiempos de ejecución y del uso de energía en las distintas pruebas y dispositivos, resaltando los mejores resultados en negrita. La tendencia general muestra que en Linux se obtiene mejor rendimiento y consumo energético. BT en ODroidXU4 es caso aislado en Linux que tarda entre 2x-3x más tiempo en ejecutarse, resultando en un consumo mayor de similares proporciones.

En cuanto a tiempos de ejecución, Rock960 es el dispositivo con un comportamiento más equilibrado entre Android y Linux, aunque las diferencias en cuanto a uso de energía son significativamente ma-



(a) HiKey970 Android

(b) HiKey970 Linux

(c) ODroidXU4 Android

(d) ODroidXU4 Linux

(e) Rock960 Android

(f) Rock960 Linux

Fig. 3. NPB BT, IS y MG, tiempo ejecución y energía relacionado con hilos

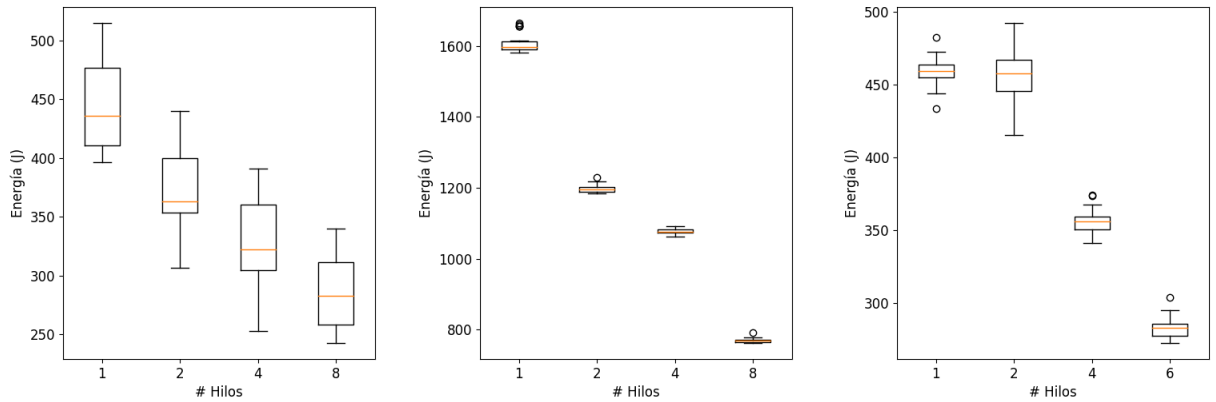
yores. Incluso en casos donde los tiempos de ejecución son muy similares utilizando el mismo número de hilos, encontramos que el consumo energético en Android puede superar ampliamente el que encontramos en Linux.

HiKey970 es el dispositivo que proporciona globalmente unos mejores resultados de rendimiento y consumo. En algunos casos está muy cerca de Rock960, especialmente en cuanto al uso de energía.

La relación entre tiempo y energía muestra que la tendencia de ejecutar menos tiempo suele conllevar un gasto menor de energía. Sin embargo, en ciertos casos existe una zona de equilibrio donde un mayor tiempo de ejecución y menor uso energético puede ser también una combinación óptima. La prueba NPB MG, que es más intensiva en memoria y comunicaciones, se ha identificado como uno de estos casos. Los binomios (19.63s, 146J) y (19.48s, 145J) en HiKey970, (39.80s, 314J) y (48.41s, 283J) en ODroidXU4, y (29.58s, 172J) y (28.92s, 167J) en Rock960, pueden considerarse un conjunto de resultados óptimos.

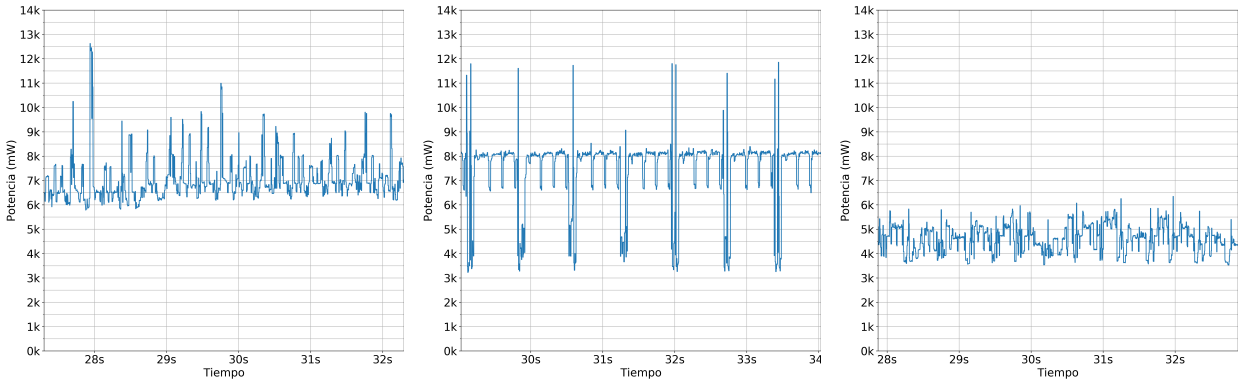
En la figura 3 se puede observar la mejora del rendimiento al aumentar el número de hilos de ejecución y el uso relativo de energía utilizada por hilo. En todos los dispositivos y sistema operativo, de forma general, se observa un incremento de rendimiento al ejecutar utilizando más hilos. En ODroidXU4 encontramos anomalías; en las figuras 3(c) y 3(d) el rendimiento se reduce al aumentar el número de hilos. Para BT el speedup es más pronunciado, pero en cualquier caso se trata de la prueba con peor desempeño. En general, encontramos aceleraciones reducidas, consiguiendo 3x como máximo y entre 1.8x-2.3x de media utilizando 8 hilos. Dado que IS y MG están limitados por la memoria, se espera este comportamiento, aunque BT es algo más computacionalmente intensivo y no se consiguió la mejora significativamente mayor prevista.

En la figura 4 se muestra los rangos intercuartílicos de la energía utilizada en los tres dispositivos. Se puede observar que el más estable es el OdroidXU4, con un uso de energía 3 veces mayor pero una variabilidad hasta 15 veces menor que los demás. La gran variación se puede explicar con el comportamiento dinámico de la potencia. La figura 5 representa una



(a) HiKey970 Linux (b) OdroidXU4 Linux (c) Rock960 Linux

Fig. 4. NPB BT, variación energía



(a) HiKey970 Linux (b) OdroidXU4 Linux (c) Rock960 Linux

Fig. 5. Ampliación de 5 segundos de NPB BT en la parte computacional

ampliación de las gráficas de potencia de 5 segundos en la parte computacional de las pruebas. Es de esperar que el tipo de trabajo que realiza la CPU influya en la gráfica de potencia dando lugar a patrones. En ODroidXU4 este patrón está bien definido, mientras que en HiKey970 y Rock960 no es evidente. La ausencia de este puede estar causada por el método de refrigeración de estos dispositivos, el cual es pasivo y basado en el control de la frecuencia del procesador. Estos detalles no son visibles en el resto de gráficas, y es importante tener los datos sobre el uso de la potencia para encontrar una explicación de la variación y valores atípicos encontrados.

Respecto a Appium hemos observado que conlleva una ligera penalización. Los servicios en segundo plano asociados aumentan el consumo comparado con el estado por defecto de entre 2-8 % para HiKey970 y Rock960. Para ODroidXU4 se incrementa entre 10-30 %. Este aumento se puede observar comparando las zonas **a** y **e** dentro de la figura 2(a), que representan, respectivamente, el consumo por defecto y una pausa durante la ejecución de la prueba.

VI. Conclusiones

En este trabajo hemos desarrollado un entorno de pruebas para facilitar la experimentación y la recogida de datos en múltiples dispositivos de manera simultánea. Se ha usado este entorno para realizar una evaluación de rendimiento y eficiencia energética para diferentes arquitecturas ARM, utilizando sistemas operativos Linux y Android. Las resultados obtenidos muestran que utilizar Android supone una carga de trabajo adicional suficientemente significativa con respecto de Linux, tanto para tiempo de ejecución como para consumo energético. Además para ambos sistemas, incrementar el número de hilos es vital para reducir la energía empleada al ejecutar los algoritmos presentados. En general, conseguir escalar las aplicaciones permite minimizar tiempo de cómputo lo que redunda en un menor consumo. En algunos casos, y con códigos no dominados por la actividad de CPU, podemos obtener mejor eficiencia energética aún cuando los tiempos de cómputo no sean los óptimos. Como trabajo futuro, incrementaremos el número de dispositivos al que realizaremos el análisis y añadiremos más software al proceso de análisis con el fin de clasificar las capacidades de las arquitecturas estudiadas.

Agradecimientos

Este trabajo ha sido cofinanciado por la CEE (FEDER) y por el Ministerio de Economía y Competitividad a través del Plan Nacional de I+D+I número TIN2016-78919-R, el Ministerio de Ciencia, Innovación y Universidades a través del contrato FPU16/00942, la Agencia Canaria de Investigación,

Innovación y Sociedad de la Información a través del proyecto ProID2017010130 y del contrato TESIS2017010134, financiado en parte por el Fondo Social Europeo (FSE) Programa Operativo Integrado de Canarias 2014-2020, Eje 3 Tema Prioritario 74 (85 %), y la Red de Computación de Altas Prestaciones sobre Arquitecturas Paralelas Heterogéneas (CAPAP-H).

Referencias

[1] "Gigabyte arm server, last accessed: May 2019," .

[2] Innovative Computing Laboratory. Univ. Tennessee, "The parallel linear algebra for scalable multi-core architectures (PLASMA) project," `http://icl.cs.utk.edu/plasma/`, 2011.

[3] Emmanuel Agullo, Jim Demmel, Jack Dongarra, Bilel Hadri, Jakub Kurzak, Julien Langou, Hatem Ltaief, Piotr Luszczek, and Stanimire Tomov, "Numerical linear algebra on emerging architectures: The PLASMA and MAGMA projects," *Journal of Physics: Conference Series*, vol. 180, no. 1, pp. 012037, 2009.

[4] A. Haidar, S. Tomov, P. Luszczek, and J. Dongarra, "Magma embedded: Towards a dense linear algebra library for energy efficient extreme computing," in *2015 IEEE High Performance Extreme Computing Conference (HPEC)*, Sep. 2015, pp. 1–6.

[5] Keren Bergman, Shekhar Borkar, Dan Campbell, William Carlson, William Dally, Monty Denneau, Paul Franzon, William Harrod, Jon Hiller, Sherman Karp, Stephen Keckler, Dean Klein, Robert Lucas, Mark Richards, Al Scarpelli, Steven Scott, Allan Snavely, Thomas Sterling, R. Stanley Williams, Katherine Yelick, Keren Bergman, Shekhar Borkar, Dan Campbell, William Carlson, William Dally, Monty Denneau, Paul Franzon, William Harrod, Jon Hiller, Stephen Keckler, Dean Klein, Peter Kogge, R. Stanley Williams, and Katherine Yelick, "Exascale computing study: Technology challenges in achieving exascale systems peter kogge, editor & study lead," 2008.

[6] Mateusz Guzek, Dzmitry Kliazovich, and Pascal Bouvry, "HEROS: energy-efficient load balancing for heterogeneous data centers," in *8th IEEE International Conference on Cloud Computing, CLOUD 2015, New York City, NY, USA, June 27 - July 2, 2015*, Calton Pu and Ajay Mohindra, Eds. 2015, pp. 742–749, IEEE.

[7] Gregor Von Laszewski, Lizhe Wang, Andrew J Younge, and Xi He, "Power-aware scheduling of virtual machines in dvfs-enabled clusters," in *2009 IEEE International Conference on Cluster Computing and Workshops*. IEEE, 2009, pp. 1–10.

[8] Chia-Ming Wu, Ruay-Shiung Chang, and Hsin-Yu Chan, "A green energy-efficient scheduling algorithm using the dvfs technique for cloud datacenters," *Future Generation Computer Systems*, vol. 37, pp. 141–147, 2014.

[9] X. Ma, Z. Deng, M. Dong, and L. Zhong, "Characterizing the performance and power consumption of 3d mobile games," *Computer*, vol. 46, no. 4, pp. 76–82, April 2013.

[10] K. Cheng and Y. Wang, "Using mobile gpu for general-purpose computing – a case study of face recognition on smartphones," in *Proceedings of 2011 International Symposium on VLSI Design, Automation and Test*, April 2011, pp. 1–4.

[11] Andrey Ignatov, Radu Timofte, William Chou, Ke Wang, Max Wu, Tim Hartley, and Luc Van Gool, "Ai benchmark: Running deep neural networks on android smartphones," in *Computer Vision – ECCV 2018 Workshops*, Laura Leal-Taixé and Stefan Roth, Eds., Cham, 2019, pp. 288–314, Springer International Publishing.

[12] Sergio Afonso, Alejandro Acosta, and Francisco Almeida, "High-performance code optimizations for mobile devices," *The Journal of Supercomputing*, vol. 75, no. 3, pp. 1382–1395, Mar 2019.

[13] Adrian Jackson, Andrew Turner, Michele Weiland, Nick Johnson, Olly Perks, and Mark Parsons, "Evaluating the arm ecosystem for high performance computing," *arXiv preprint arXiv:1904.04250*, 2019.

[14] Edson Luiz Padoin, Laércio Lima Pilla, Francieli Zanon Boito, Rodrigo Virote Kassick, Pedro Velho, and Philippe OA Navaux, "Evaluating application performance and energy consumption on hybrid cpu+ gpu architecture," *Cluster Computing*, vol. 16, no. 3, pp. 511–525, 2013.

[15] "Accelpower cape, last accessed: May 2019," .

[16] Sergio Barrachina, Maria Barreda, Sandra Catalán, Manuel F Dolz, Germán Fabregat, Rafael Mayo, and ES Quintana-Ortí, "An integrated framework for power-performance analysis of parallel scientific workloads," *Energy*, pp. 114–119, 2013.

[17] "Appium. automation for apps, last accessed: May 2019," .

[18] David H. Bailey, Eric Barszcz, John T. Barton, D. S. Browning, Robert L. Carter, Leonardo Dagum, Rod A. Fatoohi, Paul O. Frederickson, T. A. Lasinski, Robert Schreiber, Horst D. Simon, V. Venkatakrishnan, and Sisira Weeratunga, "The nas parallel benchmarks," *IJHPCA*, vol. 5, no. 3, pp. 63–73, 1991.

Implementación de algoritmos de efectos de audio en tiempo real sobre procesador digital de señal

Francisco Jiménez-Fiérrez [1], Damián Ruiz-Coll [2],
Gerardo Fernández-Escribano [1] y Pedro Cuenca [1]

Resumen— **El presente artículo presenta la implementación a bajo nivel de tres efectos de audio sobre un procesador digital de señal de propósito especifico. El artículo describe los algoritmos que implementan los distintos efectos de audio, basados en la modulación del retardo de la señal, definiendo sus características y la arquitectura para su implementación en tiempo real. De igual modo, se describe la arquitectura y prestaciones del procesador digital de señal seleccionado sobre el que se han implementado los efectos de audio programados. Por último, se proporciona la implementación a bajo nivel de los tres algoritmos propuestos y se muestran los resultados.**

Palabras clave— **Efectos de audio, DSP, procesado en tiempo real.**

I. Introducción

EN el ámbito de la interpretación musical mediante instrumentos amplificados, es común el empleo de dispositivos que procesan la señal de audio con el objetivo de aplicar algún tipo de corrección o de introducir algún tipo de efecto. La figura 1 ilustra la secuencia de funcionamiento de estos dispositivos. Previamente a la activación de la unidad de efecto, la señal de audio llega al sistema amplificador tal y como es generada en el instrumento como consecuencia de la acción del intérprete. Una vez activada y ajustados los parámetros a los valores correspondientes, la señal de audio es procesada antes de ser amplificada. El resultado de este procesado será percibido por el intérprete como alteraciones en la dinámica, timbre, tono, etc. La implementación de estas unidades se lleva a cabo de forma mecánica, mediante electrónica analógica o algoritmos digitales.

En el presente artículo se analiza y describe la implementación a bajo nivel de distintos efectos digitales de audio mediante el empleo de un procesador hardware específico denominado DSP (*digital signal processing*). El artículo ha sido organizado de la siguiente manera: la sección II describe de forma detallada las características de los efectos de audio a implementar. La sección III expone la arquitectura y funcionalidades del dispositivo empleado para ello. La sección IV expone la implementación a bajo nivel de los algoritmos y los resultados obtenidos. Finalmente, en el apartado V se muestran una serie de conclusiones y pautas para posibles líneas de trabajo futuro.

[1]Instituto de Investigación en Informática de Albacete, Universidad de Castilla-La Mancha, Albacete, España.
[2]Universidad Rey Juan Carlos, Fuenlabrada, España.

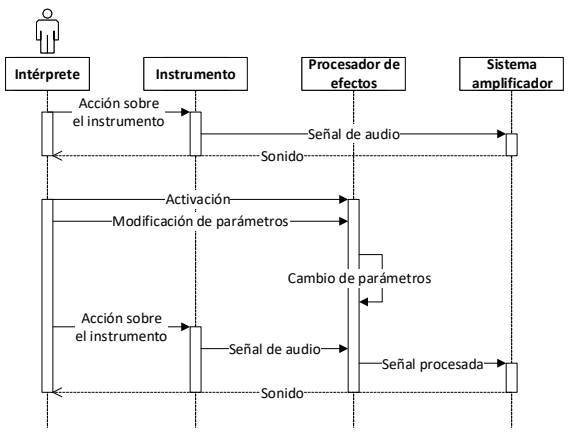


Fig. 1. Diagrama de secuencia de las unidades de efecto.

II. Efectos de modulación del retardo

Los efectos de audio recogidos bajo esta clasificación, los denominados como efectos de modulación del retardo, se basan en la combinación de la señal de audio con una réplica ligeramente retardada de la misma, siendo este retardo variable en el tiempo. Se introducen aquí los denominados efecto *chorus*, efecto *flanger* y efecto *phaser*[1][2]. Pese a tener todos ellos un fundamento similar presentan una sonoridad distinta, debido a la magnitud y las características de los retardos empleados como se detallará a continuación.

El rango de tiempos de retardo empleados en el efecto *chorus* está próximo al punto en el que los eventos temporales son disgregados por el oído. En el caso de los efectos *flanger* y *phaser*, los retardos empleados son de menor magnitud, con lo cual la copia retardada se integra con el sonido original.

La diferencia entre *phaser* y *flanger* radica en la forma en que se genera la señal retardada. En el caso del efecto *phaser* el retardo es debido al desfase producido por una serie de filtros paso todo, con lo que el retardo no es uniforme para todas las componentes de frecuencia, provocando la aparición de *notches* en el espectro de frecuencias (figura 2), los cuales se sitúan a distancia no armónica. Por el contrario, el efecto *flanger* emplea un bloque de memoria de magnitud variable para la obtención de la señal retardada, produciéndose el desplazamiento en el tiempo por igual en todas las componentes de frecuencia. Es-

te retardo uniforme da lugar a una serie de *notches* en el espectro (figura 3), situados a una distancia armónica.

A continuación se describen en detalle las características e implementación de los efectos de audio previamente mencionados.

A. Efecto chorus

El efecto *chorus*[2][3] aparece de forma natural cuando instrumentos de un mismo tipo interpretan las mismas notas de forma simultánea. Diferencias en la constitución física, afinación, o sincronización de los intérpretes, resultan en ligeras desviaciones en tono, timbre y dinámica, de forma que los sonidos no se integran en uno solo. Este efecto es común en agrupaciones corales, orquestas de cuerda e instrumentos que emplean múltiples elementos vibrantes por cada nota. La implementación del diagrama de bloques correspondiente al efecto aparece representada en la figura 4. La ecuación en diferencias correspondiente se ilustra en la expresión 1.

El retardo necesario para lograr el efecto se encuentra entre 10 ms y 25 ms. Estos valores se aproximan al valor límite en el que los eventos temporales son discernidos por el oído. Como señal moduladora del retardo es común emplear señales senoidales, triangulares o ruido de baja frecuencia. La magnitud de los retardos empleados puede dar lugra a variaciones perceptibles del tono de la señal. Estas variaciones serán más acusadas cuanto mayor sea la amplitud y frecuencia de la señal moduladora del retardo. La ecuación 2 expresa la relación existente entre los valores de frecuencia de las señales de entrada y salida para una línea de retardo modulada[3].

Existen efectos *chorus* que integran múltiples voces, como el denominado efecto *chorus* en cuadratura[2]. En este se emplea una única señal moduladora de tipo senoidal, a partir de la cual se obtienen cuatro senoides diferentes, dos senoides en cuadratura y sus correspondientes funciones opuestas. Estas cuatro funciones se emplean como punteros de acceso a la memoria de retardo, obteniendo así cuatro voces diferentes.

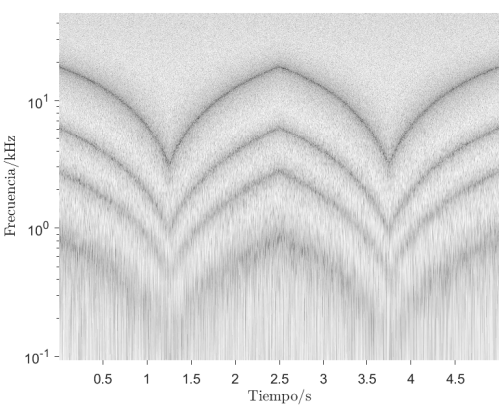


Fig. 2. Espectrograma del efecto *phaser*.

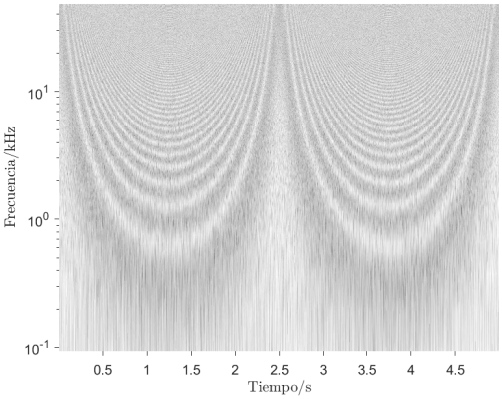


Fig. 3. Espectrograma del efecto *flanger*.

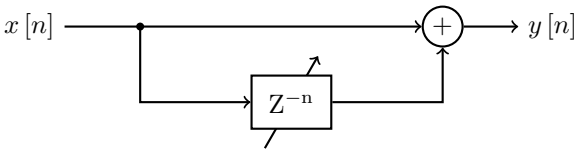


Fig. 4. Diagrama de bloques del efecto *chorus*.

B. Efecto flanger

Como ya se ha introducido anteriormente, el efecto *flanger*[4] se basa en la mezcla de una señal de audio con una copia de la misma retardada, siendo este retardo variable en el tiempo. Existen múltiples topologías del efecto dependiendo de la forma en que dichas señales se mezclan y de si existe o no realimentación.

La aplicación de este efecto densifica el sonido, dotándolo además de una cierta cualidad de movimiento, debido al desplazamiento a lo largo del espectro de las formantes generadas. Dependiendo de la intensidad del efecto se puede conseguir desde una sonoridad sutil hasta timbres fuertemente metálicos.

El término *flanger* se deriva del método primitivo de consecución de este efecto. La señal de audio

$$y\,[n] = x\,[n] + x\,[n - n_d\,(t)] \tag{1}$$

n índice de la muestra
$n_d\,(t)$ retardo en número de muestras

$$\frac{f_{\text{out}}}{f_{\text{in}}} = 1 - A_{\text{LFO}}\frac{2\pi f_{\text{LFO}}}{f_{\text{s}}}\cos\frac{2\pi f_{\text{LFO}}}{f_{\text{s}}}n \tag{2}$$

f_{out} frecuencia de la señal de salida
f_{in} frecuencia de la señal de entrada
A_{LFO} amplitud del oscilador de baja frecuencia
f_{LFO} frecuencia del oscilador de baja frecuencia
f_{s} frecuencia de muestreo
n longitud del retardo en muestras

sobre la que se pretende aplicar el efecto es grabada en dos pistas diferentes, las cuales se reproducen posteriormente de forma síncrona; siendo grabada la mezcla de ambas en una tercera. El ingeniero de sonido ralentizaba el giro de la bobina de uno de los reproductores actuando sobre el borde de su soporte (en inglés *flange*). Con esta acción el efecto se desplaza perceptualmente en una dirección, permaneciendo la pista alterada ligeramente retrasada respecto a la otra. La repetición del proceso sobre la pista restante completa el efecto.

En la década de los 70, los avances tecnológicos en dispositivos de estado sólido permitieron el desarrollo de unidades electrónicas de efecto *flanger* basadas en circuitos integrados. La implementación del diagrama de bloques de este efecto aparece representada en la figura 5. la expresión 3 representa la ecuación en diferencias correspondiente.

El tiempo de retardo empleado en la implementación de este efecto es de una magnitud menor al empleado en el efecto *chorus*, siendo normalmente inferior a los 20 ms. La suma de la señal original con su copia retardada da lugar a una estructura de filtro *comb* tipo FIR (*finite impulse response*). Esta estructura provoca unas serie de picos y valles en el espectro de la señal, los cuales se desplazan como consecuencia de la variación del retardo.

La realimentación es una característica habitual de este efecto. Parte de la señal de salida del bloque de retardo es redirigida a la entrada del mismo. El bloque de retardo realimentado constituye un filtro *comb* tipo IIR (*infinite impulse response*).

Se logran diferentes variaciones del efecto combinando las líneas original y retardada de forma aditiva o sustractiva, e invirtiendo la fase en el bucle de realimentación. Invirtiendo tanto el signo de la señal retardada como el de la realimentación se logra la variante del efecto denominada como *through-zero flanger*. La figura 6 muestra la respuesta en frecuencia de esta variante del efecto para valores extremos del retardo, con valor de realimentación del 70 %.

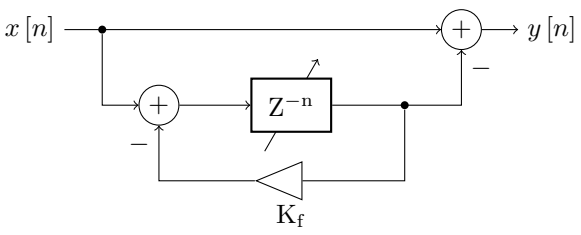


Fig. 5. Diagrama de bloques del efecto *flanger*.

$$y[n] = x[n] - x[n - n_d(t)] + K_f y[n - n_d(t)] \quad (3)$$

$n_d(t)$ retardo en número de muestras

K_f coeficiente de realimentación

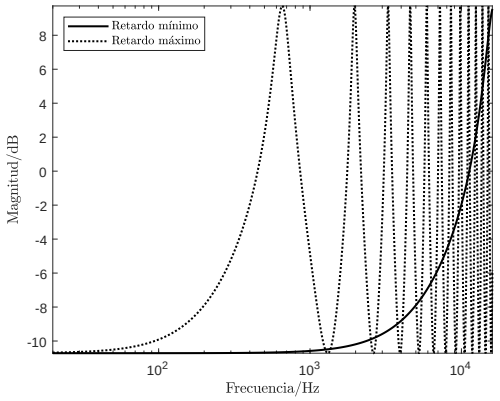


Fig. 6. Espectro de frecuencia *through-zero flanger*.

C. Efecto phaser

La sonoridad de este efecto es similar a la del efecto *flanger*, pero existen características que los hacen diferenciables. La modulación producida en este caso es más sutil, pues la incidencia sobre el espectro de la señal es menor. Por diferencias estructurales, las frecuencias afectadas son aquí equidistantes, mientras que en el caso del *flanger* se encuentran a distancia armónica.

Tanto en su implementación analógica como digital, las unidades de efecto *phaser*[5] se basan en una serie de filtros paso todo de frecuencia de corte variable, que desplazan la señal en fase; se muestra su diagrama de bloques en la figura 7. Estos filtros presentan una respuesta constante en magnitud y una respuesta en fase dependiente de la frecuencia[1]. La expresión en el dominio de la frecuencia para estos filtros viene dada por la ecuación 4. El polo de dicho filtro puede aproximarse según la ecuación 5, siendo f_c la frecuencia a la que se produce el desfase de 90°. Su valor absoluto será menor que 1 siempre y cuando el valor de la frecuencia de corte no supere el límite establecido en la ecuación 6.

El filtro paso todo anterior se representa en forma de ecuación en diferencias en la expresión 7. Esta ecuación en diferencias deberá ser implementada de forma que sea posible modificar el valor del paráme-

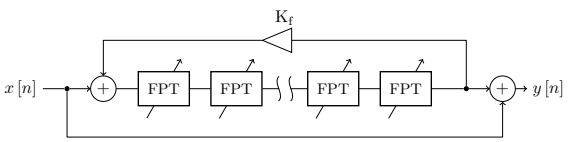


Fig. 7. Diagrama de bloques del efecto *phaser* realimentado, donde se aprecia la conexión en cascada de los sucesivos filtros paso todo (FPT).

$$A(z) = \frac{p - z^{-1}}{1 - pz^{-1}} \quad (4)$$

z variable compleja

$$p = 1 - \frac{2\pi f_c}{f_s} \tag{5}$$

f_c frecuencia de corte
f_s frecuencia de muestreo

$$f_c > \frac{f_s}{\pi} \tag{6}$$

tro p en tiempo de ejecución para la consecución del efecto.

La sucesiva conexión en cascada de filtros de este tipo da lugar a puntos del espectro en los que el desfase entre la señal original y retardada es de 180°. Es en estos puntos en los que se produce la anulación de la señal, al sumar esta con su copia retardada. Estos *notches* se sitúan a una distancia regular en frecuencia, lo que da lugar a intervalos no armónicos.

El efecto *phaser* se debe al desplazamiento de los *notches* generados a lo largo del espectro de frecuencias de la señal. Este desplazamiento se logra variando las frecuencias para las que se producen la cancelaciones de fase, para lo cual se emplean osciladores de baja frecuencia que varía los parámetros de los filtros. Normalmente la señal empleada por estos osciladores es de tipo triangular o senoidal.

Es posible intensificar el efecto realimentando la salida de la etapa de filtros hacia la entrada, como ilustra la figura 7. Esto enfatiza las frecuencias situadas entre los *notches* del espectro. La expresión 8 representa la correspondiente expresión en el dominio discreto. En la figura 8 se muestra la respuesta en frecuencia de un *phaser* realimentado.

III. Arquitectura del dispositivo

El dispositivo empleado para la implementación de los efectos es el procesador FV-1 del fabricante Spin Semiconductor[6]. Se trata de un SOC (*system on chip*) que integra un procesador digital de señales de 24 bit en coma fija, capaz de ejecutar hasta 128 instrucciones por cada muestra de audio. El dispositivo cuenta además con conversores analógico-digital y digital-analógico estéreo tipo delta-sigma, memoria para la implementación de retardos, osciladores de baja frecuencia senoidales y tipo rampa; así como operaciones logaritmo y exponencial. La representa-

$$y[n] = px[n] - x[n-1] + py[n-1] \tag{7}$$

$$Y(z) = X(z)\left[1 + \frac{\prod_{i=1}^{n} \mathrm{FPT}_i(z)}{1 - K_f \prod_{i=1}^{n} \mathrm{FPT}_i(z)}\right] \tag{8}$$

$\mathrm{FPT}_i(z)$ función de transferencia filtro paso todo
K_f coeficiente de realimentación

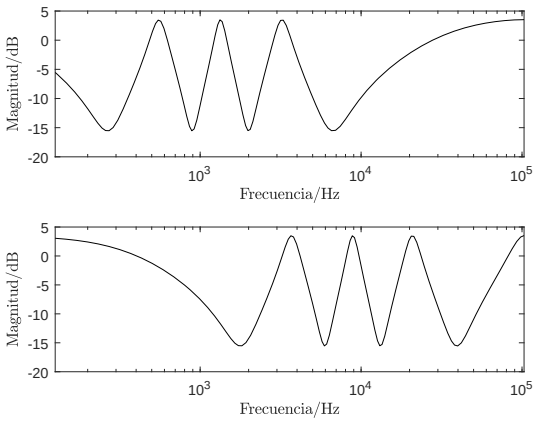


Fig. 8. Espectro de frecuencia del efecto *phaser*.

ción en diagrama de bloques de la arquitectura del dispositivo se ilustra en la figura 9.

El FV-1 es realmente un procesador en paralelo. Mientras las instrucciones de programa son ejecutadas, de forma paralela se administran las direcciones de memoria de retardo y se generan los osciladores de baja frecuencia.

El dispositivo es programado en ensamblador para una mayor eficiencia. El conjunto de instrucciones está especialmente orientado a la implementación de efectos de audio; de forma que las instrucciones del lenguaje facilitan la implementación de filtros, modulación de retardos, empleo de osciladores de baja frecuencia, etc.

El dispositivo cuenta con una serie de registros específicos que proveen acceso a elementos como conversores analógico-digital y digital-analógico, lectura de potenciómetros, etc. Adicionalmente cuenta con 32 registros de propósito general de tamaño 24 bit[7]. La memoria SRAM interna tiene un tamaño de 32k x 14 bits. Los datos son almacenados en esta memoria en un formato comprimido en coma flotante. Antes de ser utilizados en la ALU son expandidos de nuevo al formato de 24 bit en coma fija. Esta memoria está organizada en forma de buffer circular.

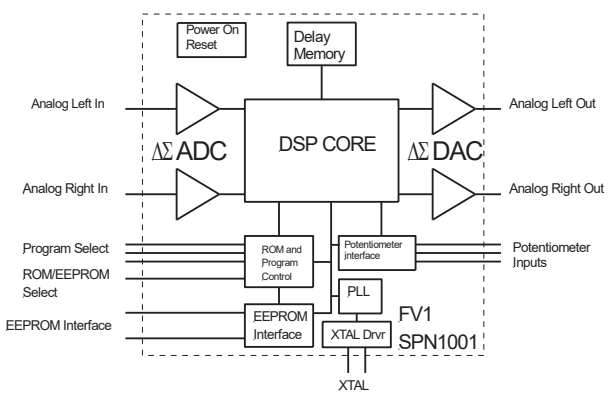


Fig. 9. Diagrama de bloques de la arquitectura del dispositivo FV-1[6].

IV. Implementación y resultados

Los efectos se implementan reproduciendo las ecuaciones en diferencias anteriormente expuestas en el lenguaje ensamblador del dispositivo FV-1. Los bloques de retardo son declarados como sectores de la memoria destinada a este efecto, la cual es gestionada en forma de *buffer* circular, de forma que los punteros de acceso a memoria son incrementados en cada ciclo de programa.

Como señales moduladoras se emplean osciladores de baja frecuencia en el caso *flanger* y *phaser*, y ruido de baja frecuencia generado a partir de los bits de menor peso de la propia señal de audio de entrada en el caso del efecto *chorus*.

En el caso de los efectos basados en línea de retardo (efecto *chorus* y efecto *flanger*), la magnitud de este es modulada variando el punto de acceso a memoria, según el valor adquirido por las señales correspondientes. El número de valores posibles que toman estas señales es superior al tamaño de los retardos empleados en número de muestras. Por esta razón, los valores de la memoria de retardo son interpolados linealmente, empleando los bits 8:22 del valor de la señal moduladora para acceder a posiciones contiguas de la memoria; mientras que los 8 bits inferiores son empleados como coeficiente de interpolación entre dichas muestras.

En el caso del efecto *phaser*, la señal moduladora obtenida a partir del oscilador de baja frecuencia es empleada para modificar el valor del parámetro p (ecuación 7), el cual determina el valor de la frecuencia de corte de los sucesivos filtros paso todo dispuestos en cascada (figura 7).

La implementación en código de los efectos expuestos, así como una demostración de su funcionamiento en forma de ficheros de audio se recogen en el siguiente repositorio `https://github.com/FranciscoJimenez-Fierrez/efectos-de-audio-en-tiempo-real-sobre-DSP`. Estos códigos están destinados a un caso de aplicación en el que el FV-1 trabaje a una frecuencia de muestreo de 32.768 kHz. A modo de ilustración de la estructura y funcionamiento de uno de estos códigos, la figura 10 muestra el diagrama de actividad del efecto *flanger* implementado.

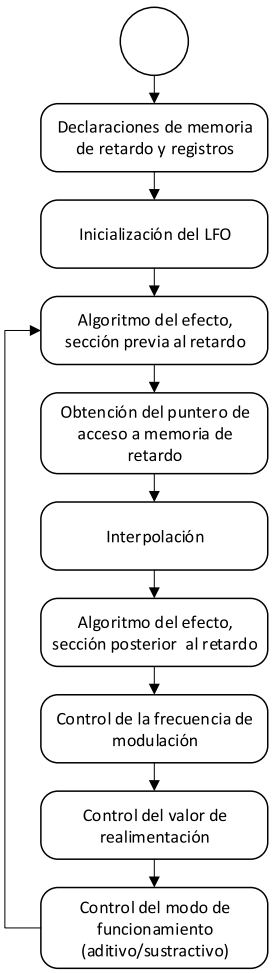


Fig. 10. Diagrama de actividad del efecto *flanger*.

V. Conclusiones y trabajo futuro

De la exposición llevada a cabo se concluye como deseable la implementación a bajo nivel de algoritmos de procesado de audio en tiempo real. Este tipo de implementación permite el control exhaustivo de las operaciones y optimiza el empleo de los recursos del sistema.

Entre las posibles líneas de trabajo futuro se encuentra el diseño mediante dispositivos reconfigurables de una versión del dispositivo FV-1 con características extendidas.

Es también interesante estudiar la posibilidad de implementar los algoritmos expuestos sobre sistemas embebidos como Raspberry Pi o Jetson Nano. La potencia de cómputo de estos dispositivos permitiría llevar a cabo procesamientos adicionales de la señal de audio.

Una práctica generalizada en la implementación de efectos digitales de audio es la búsqueda de la similitud con la implementación analógica correspondiente. Un posible desarrollo futuro con respecto a los algoritmos descritos sería el modelado de características propias de la implementación analógica de los efectos.

Agradecimientos

El presente trabajo ha sido financiado mediante la Junta de Comunidades de Castilla-La Mancha bajo el proyecto de referencia SBPLY/17/180501/000353, por el Ministerio de Ciencia, Innovación y Universidades del Gobierno de España bajo el proyecto de referencia RTI2018-098156-B-C52, y por la ayuda dirigida a grupos en el marco del Plan Propio de Investigación de la Universidad de Castilla-La Mancha con referencia 2019-GRIN-27060, todas ellas susceptibles de cofinanciación por el Fondo Europeo de Desarrollo Regional (FEDER). Además, ha sido financiado mediante las ayudas para contratos predoctorales para la formación de personal investigador en el marco del Plan Propio de Investigación de la Universidad de Castilla-La Mancha, susceptibles de cofinanciación por el Fondo Social Europeo (FSE).

Referencias

[1] Udo Zolzer, *DAFX: Digital Audio Effects*, Wiley Publishing, 2nd edition, 2011.

[2] Francisco Jiménez Fiérrez, *Trabajo Fin de Grado. Diseño de un pedal de efecto para guitarra eléctrica*, Universidad de Castilla-La Mancha, 2015.

[3] JON DATTORRO, "Effect design* part 2: Delay-line modulation and chorus," *AES: Journal of the Audio Engineering Society*, vol. 45, 10 1997.

[4] Harald Bode, "History of electronic sound modification," *Journal of the Audio Engineering Society*, vol. 32, pp. 730–739, 10 1984.

[5] Julius Smith, "An allpass approach to digital phasing and flanging," 01 1984.

[6] Spin Semiconductor, *FV-1 Reverb IC*, 2017, [Online; consultado el 15 de Abril del 2019].

[7] Spin Semiconductor, *SPINAsm & FV-1 Instruction Set*, 2008, [Online; consultado el 4 de Abril del 2019].

Arquitecturas del subsistema de memoria y almacenamiento secundario

Aceleración del Análisis de Series Temporales en el Procesador Intel Xeon Phi KNL

Iván Fernández[1], Alejandro Villegas[2], Eladio Gutiérrez[3] y Óscar Plata[4]

Resumen— **El análisis de series temporales es un campo de investigación de gran interés con innumerables aplicaciones. Recientemente, el método Matrix Profile, y particularmente una de sus implementaciones, el algoritmo SCRIMP, ha empezado a cobrar relevancia en este campo.**

En este trabajo analizamos la estructura y el rendimiento del algoritmo SCRIMP en el contexto de una arquitectura Intel Xeon Phi Knights Landing (KNL), que integra módulos de memoria HBM (High-Bandwidth Memory).

En este análisis combinamos diferentes técnicas para explotar el potencial de la arquitectura. Por un lado, explotamos la capacidad multihilo y vectorial de la arquitectura. Por otro lado, exploramos cómo ubicar los datos en la memoria para extraer el máximo rendimiento de la arquitectura de memoria híbrida disponible, haciendo uso tanto de la memoria 3D de alto ancho de banda como de la memoria convencional DRAM DDR4.

Palabras clave— **Serie Temporal, Detección de Anomalías, Paralelismo de Memoria Compartida, Memoria 3D de Alto Ancho de Banda, Intel Xeon Phi KNL.**

I. Introducción

El análisis de series temporales constituye una de las herramientas más importantes en la extracción de información sobre el comportamiento de fenómenos, con aplicabilidad en multitud de campos como como la genética [1], la sismografía [2], el transporte [3], la energía [4], etc.

Recientemente, *Matrix Profile* [5] se ha desarrollado como una potente herramienta de análisis de series temporales. Entre otras características, esta herramienta es capaz de detectar anomalías en una serie temporal. La figura 1 muestra una serie temporal y su Matrix Profile. Podemos observar un patrón periódico, aunque existe una anomalía entre los valores 100 y 120, aproximadamente.

El Matrix Profile de esta serie temporal devuelve valores bajos para la parte periódica de la serie y valores más altos donde aparece la anomalía.

Desde el punto de vista del rendimiento, la figura 2 muestra la intensidad aritmética observada en SCRIMP (número de operaciones aritméticas por byte de memoria accedido) frente al rendimiento medido en GFLOPS. En particular estas medidas han sido realizadas en el procesador multicore Intel Xeon Phi Knights Landing (KNL) [6] que integra memoria 3D de alto ancho de banda (tipo HBM) y módulos de memoria externos DRAM DDR4. El punto ① en la gráfica de la figura 2 representa el rendimiento de SCRIMP. Observamos cómo la baja intensidad aritmética hace que la aplicación no aproveche el potencial completo del procesador. En contraste, el rendimiento de SCRIMP está cerca del límite determinado por el ancho de banda de memoria de la plataforma, indicando que la memoria es el principal cuello de botella que limita el rendimiento de la aplicación.

Se pueden considerar dos opciones para mejorar el rendimiento de Matrix Profile. Una primera aproximación implica modificar el algoritmo para incrementar su intensidad aritmética. Esto podría mover el rendimiento al punto ② en la figura 2, pero estaría aún limitado por el ancho de banda de memoria. Si la intensidad aritmética se mejora lo suficiente, la aplicación podría incluso alcanzar el pico de rendimiento de la plataforma. No obstante, modificar el algoritmo requiere un gran esfuerzo y podría incluso no ser posible.

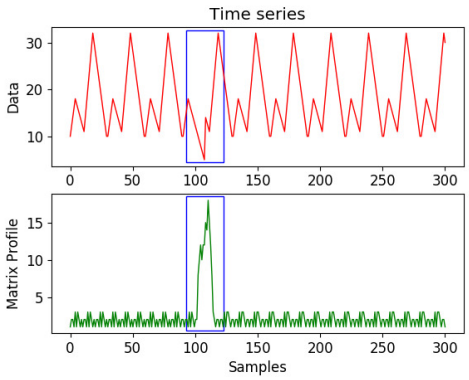


Fig. 1
Una serie con una anomalía y su Matrix Profile

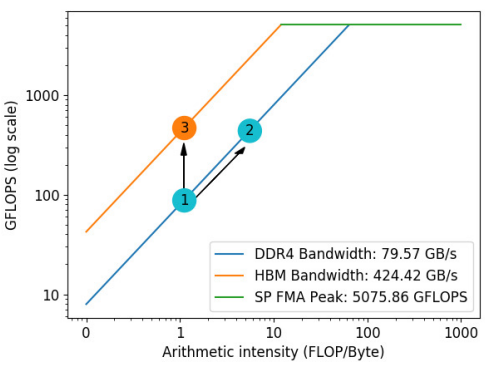


Fig. 2
Roofline del procesador KNL para SCRIMP

[1]Dpto. de Arquitectura de Computadores, Univ. Málaga, e-mail: ivanfv@uma.es.
[2]Dpto. de Arquitectura de Computadores, Univ. Málaga, e-mail: avillegas@uma.es.
[3]Dpto. de Arquitectura de Computadores, Univ. Málaga, e-mail: eladio@ac.uma.es.
[4]Dpto. de Arquitectura de Computadores, Univ. Málaga, e-mail: oplata@uma.es.

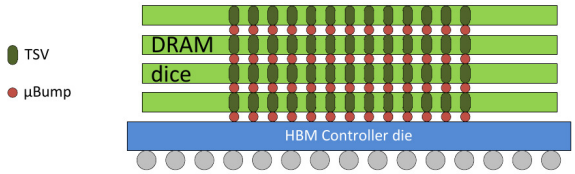


Fig. 3
Estructura de una memoria HBM

Otra aproximación consiste en explotar la memoria HBM integrada en el procesador. HBM es una inferfaz RAM de alto rendimiento para memorias 3D [7]. Como se muestra en la figura 3, esta organización de memoria consiste de varias capas de silicio conectadas verticalmente con TSVs (*Through-Silicon Vias*) y *microbumps*. Con esta organización, la memoria HBM ofrece un ancho de banda pico de unos 450 GB/s, en contraste con los 80 GB/s ofrecidos por la memoria DDR4. Usando adecuadamente la memoria HBM, combinada o no con la memoria DDR4, y asumiendo que la intensidad aritmética se mantiene inalterada, el rendimiento del algoritmo se puede llevar al punto ③ de la figura 2.

Este trabajo se centra en acelerar el algoritmo SCRIMP, cuyo cuello de botella principal es el ancho de banda de memoria, en el procesador KNL, que combina las tecnologías de memoria tipo HBM y DDR4. Nuestro análisis implica el aprovechamiento máximo de los recursos computacionales, tanto de los núcleos como de la jerarquía de memoria.

II. Antecedentes

A. Matrix Profile y SCRIMP

En esta subsección se describe sucintamente la estructura Matrix Profile [5], y el algoritmo SCRIMP [8].

Una *serie temporal* T es una secuencia de valores reales de longitud n. Llamamos t_i al i-ésimo valor de T. Una *subsecuencia* $T_{i,m}$ es un subconjunto de T empezando de la posición i y de longitud m. Dada una longitud m, un *Matrix Profile* P de una serie temporal T es un vector conteniendo la distancia entre cada subsecuencia de T y su subsecuencia más cercana (es decir, $P_i = d_{i,j}$, si $T_{j,m}$ es la subsecuencia más cercana a $T_{i,m}$).

El *Matrix Profile Index* I es un vector de índices donde $I_i = j$ si $P_i = d_{i,j}$. P contiene las mínimas distancias de las subsecuencias de T, mientras que I es un vector de índices a la localización de esas subsecuencias.

Para computar el Matrix Profile P necesitamos calcular la distancia $d_{i,j}$ para cada subsecuencia de T. Como las subsecuencias vecinas de $T_{i,m}$ son similares a esa subsecuencia, son ignoradas definiendo una zona de exclusión. La figura 4 muestra dos subsecuencias de una serie T y la zona de exclusión para una de ellas.

El objetivo del Matrix Profile es calcular la distancia para cada par de subsecuencias $T_{i,m}$ y $T_{j,m}$. Como se muestra en la figura 5, esas distancias pueden ser organizadas como una matriz.

No obstante, el tamaño de esa matriz es enorme para series temporales largas, y puede no ser posible almacenarla en memoria. Por ese motivo, los algoritmos basados en Matrix Profile solo guardan el vector de distancias mínimas P y sus índices en I.

El algoritmo SCRIMP computa la distancia euclídea $d_{i,j}$ de dos subsecuencias como:

$$d_{i,j} = \sqrt{2m\left(1 - \frac{Q_{i,j} - \mu_i\mu_j}{m\sigma_i\sigma_j}\right)} \qquad (1)$$

En esta expresión, $Q_{i,j}$ es el producto escalar de las subsecuencias i y j, y puede ser calculada directamente, o si se conoce un producto escalar anterior, mediante la expresión $Q_{i,j} = Q_{i-1,j-1} - t_{i-1}t_{j-1} + t_{i+m-1}t_{j+m-1}$ (ver [9]). Esto implica que el cálculo de $Q_{i,j}$ crea una dependencia entre los elementos de la misma diagonal (figura 5). En este sentido, el producto escalar solo tiene que ser calculado para la primera fila (o columna) de la tabla, y los valores restantes pueden ser calculados usando los valores previos. Los símbolos μ_k y σ_k representan la media y la desviación estándar de la subsecuencia $T_{k,m}$, respectivamente, dado un tamaño de ventana m. Esos valores pueden ser calculados de antemano siguiendo la técnica introducida en [10].

La figura 6 muestra un pseudo código del algoritmo SCRIMP. Inicialmente, la media y la desviación estándar se precalculan (línea 2) además de inicializar el Matrix Profile y el vector de índices (línea 3). A continuación se calculan las diagonales (ver figura 5) (líneas 4–19).

La variable `Diagonals` es el conjunto de todas las diagonales que se necesitan para calcular P y I. Estas diagonales pueden elegirse aleatoriamente, permitiendo soluciones parciales o en orden, que permiten mayores optimizaciones [8]. Ambas opciones son exploradas en este trabajo. Cuando se procesa un elemento de la diagonal, necesitamos o bien computar el producto escalar si se trata del primer elemento (línea 7), o usar los resultados previos (línea 9). Una vez calculada la distancia (línea 11), se reemplaza la distancia previa si es menor (líneas 12–14). Como el Matrix Profile guarda la distancia $d_{i,j}$ entre dos subsecuencias $T_{i,m}$ y $T_{j,m}$, una segunda condición comprueba si $d_{j,i}$ necesita ser actualizada (líneas 15–17). Esta comprobación es necesaria porque no hay garantías de reciprocidad entre dos subsecuencias.

B. Intel Xeon Phi KNL

Como plataforma de experimentación hemos usado un SuperMicro Superserver 5038K-i [11], con un procesador Intel Xeon Phi 7210 (KNL) [12]. Esta arquitectura ofrece la combinación de tecnologías de memoria HBM con DDR4, lo que permite experimentar con la ubicación de datos en cualquiera de ellas, según sea más conveniente [13]. El procesador KNL incluye 64 núcleos Airmont (Atom) con soporte de 256 contextos hardware de ejecución (*Hyperthrea-*

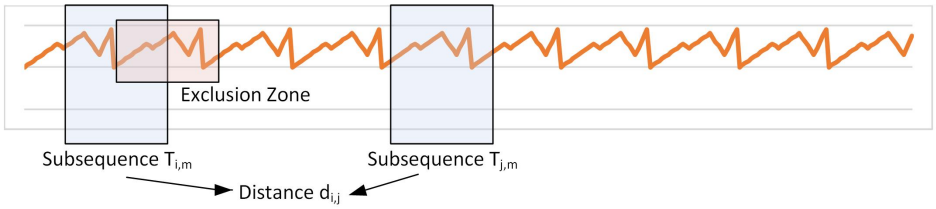


Fig. 4
Dos subsecuencias $T_{i,m}$ y $T_{i,m}$ de una serie dada T. La zona de exclusión de $T_{i,m}$ se ignora

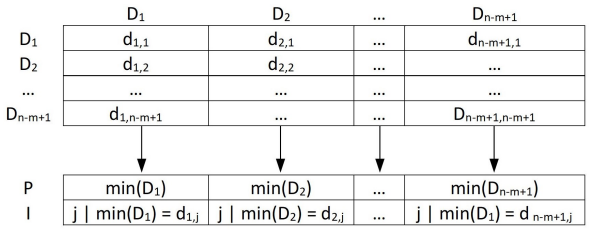


Fig. 5
Cálculo de P y sus índices I de la matriz de distancias

```
1  //Algoritmo SCRIMP
2  μ, σ = precalculateMeanSTD(T,m);
3  initialize(P, inf); initialize(I,0);
4  for k in Diagonals
5     for i=1 to length(k)
6        if i=1
7           q = dotProduct(T_{i,m}, T_{diag,m});
8        else
9           q = q - t_{i-1}t_{j-1} + t_{i+m-1}t_{j+m-1};
10       endif
11       d = distance(q, μ_i, σ_i, μ_{i+k-1}, σ_{i+k-1});
12       if d < P_i
13          P_i = d; I_i = i + k - 1;
14       endif
15       if d < P_{i+k-1}
16          P_{i+k-1} = d; I_{i+k-1} = i;
17       endif
18    endfor
19 endfor
20 return P,I;
```

Fig. 6
Pseudo código de SCRIMP

ding), esto es 4 contextos por núcleo. Además contiene una memoria 3D tipo HBM de 16GB integrada en el chip del procesador, ofreciendo un mayor ancho de banda que los módulos DDR4 externos. La memoria 3D se basa en DRAM multicanal (MCDRAM) y consta de cuatro bancos con un ancho de banda máximo agregado de más de 450 GB/s. La memoria DDR4, cuya capacidad es de 192 GB, proporciona 6 canales con un ancho de banda pico de 115.2 GB/s.

Se puede explotar paralelismo multihilo y vectorial, usando las extensiones SIMD con tecnología AVX-512. Estas extensiones permiten el cómputo vectorial de 16 operaciones de coma flotante de precisión simple u 8 de doble precisión, mediante dos procesadores vectoriales fuera de orden (VPU) disponibles por cada *tile*. El rendimiento máximo teórico del procesador KNL es de 6 TFLOPS en precisión simple y 3 TFLOPS en precisión doble.

III. Optimización de SCRIMP en KNL

SCRIMP es altamente paralelizable debido a que el cómputo de las diagonales se puede realizar independientemente. La forma más sencilla de acelerar el algoritmo en un sistema multi-core consiste en distribuir las diagonales entre los diferentes hilos de ejecución. No obstante, como las diagonales tienen diferente longitud, esa distribución puede llevar a un desbalanceo de carga. Una forma de solucionar este inconveniente consiste en crear una lista de diagonales para ser procesadas, y cada vez que un hilo acaba de computar una diagonal toma una nueva de la lista. De esta manera, los hilos están ocupados la mayor parte del tiempo. Puesto que existe la posibilidad de que varios hilos actualicen la misma posición del Profile, debemos establecer un mecanismo que resuelva los conflictos. En un primer lugar, desarrollamos una versión basada en operaciones atómicas, que debido al pobre rendimiento obtenido sólo mostraremos sus resultados por comparación. La solución que obtiene mejores resultados está basada en la privatización de los datos, que describimos en la siguiente subsección. También hemos realizado diversas optimizaciones en la implementación de SCRIMP para aumentar la intensidad aritmética y aprovechar las capacidades vectoriales de la plataforma.

A. Actualización de P y I

Las posibles actualizaciones concurrentes a las estructuras de datos P y I se pueden resolver o bien mediante exclusión mutua (*atomics*) o mediante privatización de los datos críticos. El rendimiento del primer enfoque está fuertemente influenciado por la sincronización. Además, aunque en la privatización la sincronización no es necesaria, necesita una cantidad extra de memoria.

Privatización del Profile y del vector de índices. Esta implementación está basada en la privatización de los accesos a las estructuras de datos compartidas. Nuestro objetivo es evitar el uso de métodos de exclusión mutua, en busca del máximo rendimiento. Esto se consigue expandiendo [14] el Profile, creando una réplica (fila) por hilo, como se muestra en la figura 7. Esto es computacionalmente seguro ya que la operación realizada por iteración es conmutativa y asociativa, es decir, es un bucle de reducción [15]. Aun siendo compartida, cada réplica puede ser vista como un almacenamiento privado durante el cálculo del Matrix Profile, de manera que cada hilo computa su Matrix Profile privado, guardando los resultados parciales en él. La única sincronización necesaria es una barrera para esperar que

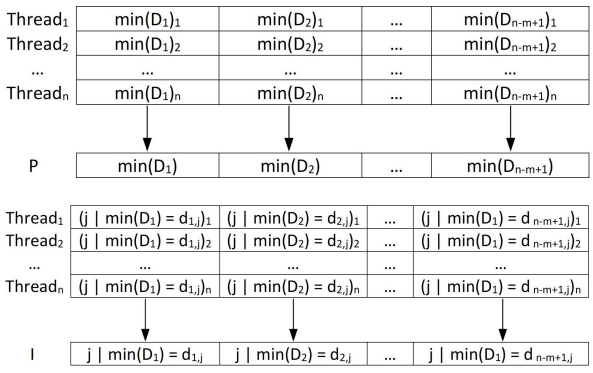


Fig. 7
Estructura expandida para P (arriba) e I (abajo)

```
1 /* Expandiendo estructuras privadas */
2 double profile_exp[nThreads][ProfileLength];
3 int profileIndex_exp[nThreads][ProfileLength];
4
5 void SCRIMP()
6 {
7   #pragma omp for schedule(dynamic)
8   for (j=0; j<numDiags; j++) {
9     for (i=j; i<diagLength; i++) {
10      /* distance is calculated here */
11      UpdateProfile(distance,i,j,threadID);
12    }
13  }
14  #pragma omp barrier
15  FinalReduction();
16 }
17
18 void UpdateProfile(distance, i, j, threadID)
19 {
20   id = threadID;
21   if (distance < profile_exp[id][j]) {
22     profile_exp[id][j] = distance;
23     profileIndex_exp[id][j] = i;
24   }
25   if (distance < profile_exp[id][i]) {
26     profile_exp[id][i] = distance;
27     profileIndex_exp[id][i] = j;
28   }
29 }
30
31 void FinalReduction()
32 {
33   #pragma omp for schedule(static)
34   for (col=0; col<ProfileLength; col++) {
35     for(row=0; row<numThreads; row++) {
36       if(profile_exp[row][col] < min_distance)
            {
37       min_distance = profile_exp[row][col];
38       min_index=profileIndex_exp[row][col];
39       }
40     }
41     profile[col] = min_distance;
42     profileIndex[col] = min_index;
43   }
44 }
```

Fig. 8
Privatización del Matrix Profile

todos los hilos acaben su computación privada. Una vez llegado a ese punto, realizamos una reducción por columnas en paralelo. Obsérvese que para el caso del vector de índices se puede realizar un procedimiento análogo (figura 7). La expansión involucra declarar dos matrices que pueden ser accedidas por los hilos: una correspondiente a las distancias y otra correspondiente a los índices, como muestra el código en la figura 8. Cada hilo actualiza las nuevas distancias en el Profile correspondientes a una fila basado en su identificador de hilo (líneas 20–28), y la reducción final se realiza en paralelo sin sincronización (líneas 33–43).

```
1 while(j < (ProfileLength - ARIT_FACT))
2 {
3   #pragma omp simd
4   for(int k=0; k<ARIT_FACT; k++) {
5       Q[k] = /* Q value based on j*/;
6   }
7
8   #pragma unroll (ARIT_FACT - 1)
9   for(int k=1; k<ARIT_FACT; k++) {
10      Q[k] += Q[k-1];
11  }
12
13  #pragma omp simd
14  for(int k=0; k<ARIT_FACT; k++) {
15      /* ξ= some calculations based on j */
16      distances[k] =  Q[k] + ξ;
17  }
18  j+=ARIT_FACT;
19 }
```

Fig. 9
Desenrollado de los bucles internos y vectorización

B. Aumento de la Intensidad Aritmética

Además de la paralelización del algoritmo, es posible reducir el tiempo de de ejecución aún más usando el soporte vectorial disponible en la plataforma KNL. Para este cometido, empaquetamos varios cálculos de distancias y actualizaciones en grupos, en vez de calcularlas individualmente. En particular, optimizamos el cálculo del producto escalar $Q_{i,j}$ de la ecuación (1). Debe tenerse en cuenta que hay dependencias de datos entre elementos consecutivos de una diagonal. Consecuentemente, esta parte de la computación será llevada a cabo en un bucle desenrollado y no vectorizado, y posteriormente usada para calcular la parte no dependiente de la computación que puede ser vectorizada dentro de otro bucle. La figura 9 muestra cómo se lleva a cabo la computación. En primer lugar, precalculamos el componente del valor que no tiene dependencias de datos (líneas 3–6) en un bucle vectorizado. Después de eso, podemos calcular los valores, teniendo en cuenta las dependencias y usando un bucle desenrollado (líneas 8–11). Finalmente, los valores de distancia derivan de los valores precalculados de nuevo en un bucle vectorizado (líneas 13–17).

Nótese que la constante *ARIT_FACT* es dependiente de la arquitectura y guía al compilador cuando genera el código máquina. En nuestro caso, esta constante está definida con un valor de 8, para aprovechar las instrucciones vectoriales de 512 bits (tenemos 8 números en coma flotante de doble precisión guardados en un mismo registro vectorial).

C. Política de Ubicación en Memoria

Con el objetivo de aprovechar el máximo ancho de banda de memoria disponible (HBM más DDR4), nuestro enfoque aloja las variables más frecuentemente accedidas en el espacio HBM, donde está disponible el mayor ancho de banda. Concretamente, HBM almacena tanto las medias y desviaciones estándar computadas para cada subsecuencia (parámetros μ y σ en la ecuación (1)), como las estructuras privatizadas de distancias mínimas e índices. No obstante, la serie original y el vector de distancias final junto a los índices se alojan en la memo-

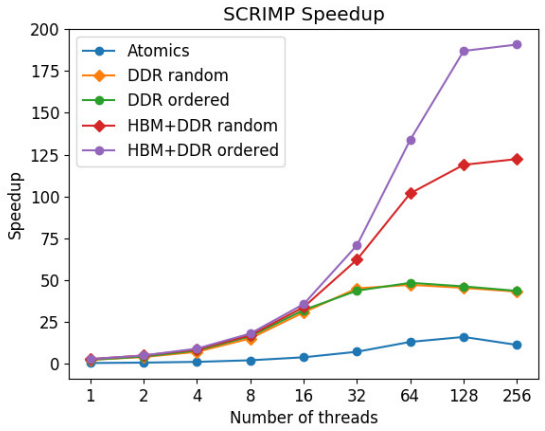


Fig. 10
Aceleración de SCRIMP en el Xeon Phi KNL

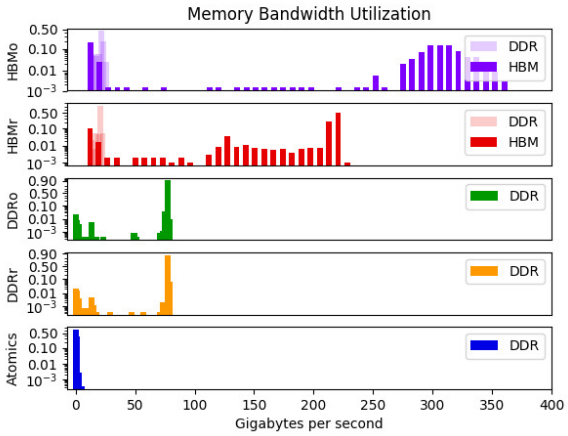


Fig. 11
Utilización del ancho de banda de memoria por SCRIMP en el Xeon Phi KNL

ria DDR4. Usando este enfoque, la memoria DDR4 sirve la serie temporal a los hilos mientras el cálculo de las diagonales es llevado a cabo (solo lecturas); después de eso, la memoria DDR4 es usada otra vez para guardar los resultados finales obtenidos en la última fase de reducción.

IV. Evaluación Experimental

A. Entorno Experimental

Los experimentos se han llevado a cabo sobre el procesador Intel Xeon Phi 7210 descrito en la sección II-B. Los códigos han sido compilados con el compilador Intel C++ v.18.0.2, con optimizaciones `-O3 -xHost` para generar el código con el mejor conjunto de instrucciones disponible (en este caso, AVX-512 cuando sea posible). Los resultados presentados están basados en el valor promedio de 10 ejecuciones.

B. Resultados de Aceleración

En primer lugar, hemos llevado a cabo los experimentos relacionados con la aceleración y uso de ancho de banda con una longitud fija tanto de serie temporal como de ventana, usando la implementación descrita en la sección III además de la versión atómica. Hemos medido el uso de ancho de banda variando el número de hilos en la implementación que nos proporciona los mejores resultados desde el punto de vista del rendimiento. Finalmente, hemos ejecutado nuestra implementación basada en en privatización de estructuras variando la longitud de la serie temporal, el tamaño de la ventana y la política de ubicación en memoria, proporcionando una comparación con los trabajos previos.

En la figura 10 se muestran los distintos resultados obtenidos. Este caso corresponde a una serie temporal de 2^{17} elementos y un tamaño de ventana de 1024.

Además de la versión atómica, hemos evaluado cuatro versiones de la versión privatizada. Por un lado, hemos probado tanto la combinación de la memoria HBM más la DDR4 como usar la memoria DDR4 solamente. Por otro lado, cuando computamos las diagonales, consideramos tanto orden aleatorio como secuencial.

La aceleración observada para estas configuraciones se muestra en la figura 10. Estableciendo un orden aleatorio da la ventaja de la propiedad *anytime*, que permite al usuario parar la computación en cualquier parte de ella, obteniendo una versión parcial (no exacta) de los resultados. La desventaja de ese enfoque es que es esperado un menor reuso de datos en las cachés, y esto explica por qué computar las diagonales en orden secuencial brinda un mejor rendimiento.

Más aún, los dos casos en los que solo se usa DDR4 (tanto orden aleatorio como secuencial) obtienen un rendimiento similar, creciendo significativamente hasta los 16 hilos y empezando a decrecer a partir de 64. A partir de ese punto, el ancho de banda DDR4 no es suficiente para servir a los hilos los datos que necesitan.

Por contra, los casos en los que se usa HBM mantienen el crecimiento con el número de hilos, y a partir de 16 hilos quedan bastante lejos de los casos con sólo DDR4. También ha de considerarse que el número de núcleos físicos disponibles es 64, y emplear más hilos implica el uso de *hyperthreading*. Este hecho resulta en una menor incremento del rendimiento si se compara con el caso de un hilo por núcleo. Obsérvese que la ejecución de un solo hilo es más rápida que la secuencial, debido al incremento en la intensidad aritmética obtenida a través del uso de las instrucciones AVX-512 en la mayor parte de los cálculos, como se explica en la sección III-B.

C. Resultados de Ancho de Banda de Memoria

El uso del ancho de banda de memoria fue medido con la herramienta Intel VTune [16]. La figura 11 muestra los tiempos de ejecución normalizados para cada uso de ancho de banda dada una serie temporal aleatoria de 2^{18} elementos, que permite resultados más precisos con esta herramienta. Hemos usado un tamaño de ventana de 1024, 256 hilos y las mismas cinco configuraciones que en la subsección anterior.

TABLA I
SCRIMP usando solo DDR4 en orden aleatorio

m	2^{17}	2^{18}	2^{19}	2^{20}	2^{21}
1024	5.65s	24.18s	119.70s	579.01s	2599.26s
2048	5.51s	23.84s	119.51s	590.99s	2615.70s
4096	5.32s	23.42s	118.71s	592.58s	2622.94s
8192	4.92s	22.54s	116.38s	586.66s	2637.59s
16384	4.21s	20.66s	111.77s	577.15s	2611.05s

TABLA II
SCRIMP usando solo DDR4 en orden secuencial

m	2^{17}	2^{18}	2^{19}	2^{20}	2^{21}
1024	5.62s	23.49s	104.26s	468.75s	2063.70s
2048	5.51s	23.29s	104.09s	472.22s	2100.94s
4096	5.32s	22.82s	103.97s	473.78s	2148.62s
8192	4.93s	21.94s	101.25s	474.86s	2088.82s
16384	4.24s	20.24s	97.98s	464.04s	2072.78s

TABLA III
SCRIMP usando HBM más DDR4 en orden aleatorio

m	2^{17}	2^{18}	2^{19}	2^{20}	2^{21}
1024	1.99s	8.38s	37.37s	153.61s	568.52s
2048	2.17s	9.22s	40.72s	169.43s	614.55s
4096	2.17s	9.35s	41.03s	173.04s	631.99s
8192	2.03s	9.07s	40.43s	173.02s	639.72s
16384	1.82s	8.48s	39.14s	167.54s	637.03s

TABLA IV
SCRIMP usando HBM más DDR4 en orden secuencial

m	2^{17}	2^{18}	2^{19}	2^{20}	2^{21}
1024	1.28s	4.88s	20.71s	88.19s	380.82s
2048	1.27s	4.86s	21.13s	90.09s	397.05s
4096	1.24s	4.88s	20.77s	90.17s	406.58s
8192	1.18s	4.79s	20.46s	91.64s	402.10s
16384	1.03s	4.48s	19.88s	90.43s	411.40s

En la versión de atómicos el ancho de banda usado es muy pequeño, puesto que la mayor parte del tiempo los hilos están compitiendo por los *locks*. Los dos casos correspondientes al uso exclusivo de la memoria DDR4 presentan un alto uso del ancho de banda teniendo en cuenta el máximo de este tipo de memoria. Este máximo no es suficiente para que el orden secuencial, al tomar las diagonales, sea más ventajoso.

Con respecto a las implementaciones basadas en la combinación de HBM con DDR4, podemos observar que, mientras un orden aleatorio de las diagonales usa un alto ancho de banda de memoria, el orden secuencial alcanza el máximo para este tipo de memoria la mayor parte del tiempo.

D. Sensibilidad a la Longitud de la Serie Temporal y el Tamaño de la Ventana

Las tablas I a IV muestran la sensibilidad de las implementaciones cuando se varía la longitud de la serie temporal y el tamaño de la ventana. Todos estos tests usan el número de hilos que proporcionan el menor rendimiento. Para llevar a cabo estos experimentos, hemos definido series temporales con tamaños representativos. En particular, usamos series temporales de 2^{17}, 2^{18}, 2^{19}, 2^{20} y 2^{21} elementos, puesto que son los valores más usados en la literatura [8], además de los tamaños de ventana más comunes (m).

La tabla I muestra los resultados de ejecutar SCRIMP con DDR4 exclusivamente y estableciendo un orden aleatorio para las diagonales. Como sugieren los resultados, no hay una clara correlación entre el tamaño de la ventana y el tiempo de ejecución, pero depende del número de elementos que caben en las cachés. En caso de que establezcamos un orden secuencial para computar las diagonales, obtenemos tiempos de ejecución similares a los del caso anterior, como muestra la tabla II. No hay beneficios significativos en el rendimiento para series temporales cortas, pero para más largas podemos obtener hasta un 25 % de mejora en los tiempos de ejecución.

También hemos evaluado la combinación de HBM con DDR4. Si usamos la memoria HBM para alojar las variables más usadas y un orden aleatorio para las diagonales, obtenemos desde un 2.8x hasta un 4.6x de aceleración con respecto al uso exclusivo de la DDR4, dependiendo del tamaño de la ventana y de la longitud de la serie temporal. Las series más largas obtienen más beneficio del uso de la HBM que las más pequeñas, como se muestra en la tabla III. Los experimentos usando el orden secuencial para las diagonales y la combinación de las memorias HBM más la DDR4 se muestran en la tabla IV. En este caso, obtenemos hasta un 58 % de mejores tiempos de ejecución que la misma configuración con orden aleatorios de las diagonales.

V. Conclusiones

En este trabajo hemos presentado una implementación eficiente del algoritmo SCRIMP de Matrix Profile en una arquitectura Xeon Phi KNL. En esta implementación explotamos los múltiples núcleos, la vectorización y el uso agregado de ancho de banda de memorias HBM y DDR4.

Hemos probado dos enfoques diferentes. En primer lugar, hemos propuesto la paralelización de las diagonales en SCRIMP, distribuyéndolas dinámicamente en los núcleos. En segundo lugar, mediante privatización reducimos la presión de sincronización entre hilos, mejorando así la escalabilidad. Además, hemos incrementado la intensidad aritmética de nuestras implementaciones usando operaciones vectoriales. Finalmente, hemos propuesto la distribución de los datos en los espacios DDR4 y HBM, alojando los datos privatizados y los más frecuentemente usados en la HBM y los datos compartidos de solo lectura en la DDR4.

Los experimentos muestran mejoras en el rendimiento en hasta 190× con respecto a la ejecución secuencial usando un Xeon Phi 7210 (KNL) de 64 núcleos. Finalmente, la implementación usando tanto HBM como DDR4 es capaz de ejecutarse hasta 5x más rápido que la solución basada solo en DDR4, probando los beneficios del uso de HBM para problemas limitados por el ancho de banda.

Agradecimientos

Este trabajo se ha realizado gracias a la financiación ofrecida por los proyectos TIN2016-80920-R del Ministerio de Economía, Industria y Competitividad, y P12-TIC-1470 de la Junta de Andalucía.

Referencias

[1] Zu-Guo Yu and Vo Anh, "Time series model based on global structure of complete genome," *Chaos, Solitons & Fractals*, vol. 12, no. 10, pp. 1827–1834, 2001.

[2] Clara E Yoon, Ossian O'Reilly, Karianne J Bergen, and Gregory C Beroza, "Earthquake detection through computationally efficient similarity search," *Science advances*, vol. 1, no. 11, pp. e1501057, 2015.

[3] Li Li, Xiaonan Su, Yi Zhang, Yuetong Lin, and Zhiheng Li, "Trend modeling for traffic time series analysis: An integrated study," *IEEE Transactions on Intelligent Transportation Systems*, vol. 16, no. 6, pp. 3430–3439, 2015.

[4] "REFIT: Smart homes and energy demand reduction," `www.refitsmarthomes.org/index.php/data`, Accedido 2 Mayo 2019.

[5] Chin-Chia Michael Yeh, Yan Zhu, Liudmila Ulanova, Nurjahan Begum, Yifei Ding, Hoang Anh Dau, Diego Furtado Silva, Abdullah Mueen, and Eamonn Keogh, "Matrix Profile I: All pairs similarity joins for time series: A unifying view that includes motifs, discords and shapelets," in *16th IEEE International Conference on Data Mining (ICDM)*, 2016, pp. 1317–1322.

[6] James Jeffers, James Reinders, and Avinash Sodani, *Intel Xeon Phi Processor High Performance Programming: Knights Landing Edition*, Morgan Kaufmann, 2016.

[7] Gabriel H Loh, "3D-stacked memory architectures for multi-core processors," *ACM SIGARCH Computer Architecture News*, vol. 36, no. 3, pp. 453–464, 2008.

[8] Yan Zhu, Chin-Chia Michael Yeh, Zachary Zimmerman, Kaveh Kamgar, and Eamonn Keogh, "Matrix Profile XI: SCRIMP++: Time series motif discovery at interactive speeds," in *18th IEEE International Conference on Data Mining (ICDM)*, 2018.

[9] Yan Zhu, Zachary Zimmerman, Nader Shakibay Senobari, Chin-Chia Michael Yeh, Gareth Funning, Abdullah Mueen, Philip Brisk, and Eamonn Keogh, "Matrix Profile II: Exploiting a novel algorithm and GPUs to break the one hundred million barrier for time series motifs and joins," in *16th IEEE International Conference on Data Mining (ICDM)*, 2016, pp. 739–748.

[10] Thanawin Rakthanmanon, Bilson Campana, Abdullah Mueen, Gustavo Batista, Brandon Westover, Qiang Zhu, Jesin Zakaria, and Eamonn Keogh, "Searching and mining trillions of time series subsequences under dynamic time warping," in *18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2012, pp. 262–270.

[11] "Superserver 5038k-i specs.," `www.supermicro.com/products/system/tower/5038/SYS-5038K-I.cfm`, Accedido 2 Mayo 2019.

[12] "Intel Xeon Phi 7210 specs.," `https://ark.intel.com/products/94033`, Accedido 2 Mayo 2019.

[13] Avinash Sodani, Roger Gramunt, Jesus Corbal, Ho-Seop Kim, Krishna Vinod, Sundaram Chinthamani, Steven Hutsell, Rajat Agarwal, and Yen-Chen Liu, "Knights Landing: Second-generation Intel Xeon Phi product," *IEEE Micro*, vol. 36, no. 2, pp. 34–46, 2016.

[14] P. Feautrier, "Array expansion," in *2nd ACM International Conference on Supercomputing (ICS)*, 1988, pp. 429–441.

[15] E. Gutiérrez, O. Plata, and E. L. Zapata, "A compiler method for the parallel execution of irregular reductions in scalable shared memory multiprocessors," in *14th International Conference on Supercomputing (ICS)*, 2000, pp. 78–87.

[16] "Intel VTune website," `https://software.intel.com/en-us/vtune`, Accedido 2 Mayo 2019.

Particionado eficiente de cache en cluster para mejorar la justicia en procesadores multicore comerciales

Adrián García, Juan Carlos Sáez, Fernando Castro y Manuel Prieto [1]

Resumen—
Los procesadores multicore están ampliamente presentes en sistemas de cómputo de diferentes segmentos del mercado. A pesar de sus beneficios, la contención que aparece de forma natural cuando múltiples aplicaciones compiten por el uso de los recursos compartidos entre los cores –como la caché de último nivel (LLC)– puede causar una degradación sustancial en el rendimiento global. Se ha demostrado que asignar aplicaciones de la carga de trabajo a particiones disjuntas de la LLC, posiblemente de diferentes tamaños, ayuda a mitigar los efectos de la contención en recursos compartidos.

En este artículo proponemos LFOC, una estrategia de particionado en cluster que intenta proporcionar justicia y, al mismo tiempo, mantener un rendimiento global del sistema aceptable. LFOC utiliza la tecnología Intel CAT, que permite dividir la LLC en particiones. Para alcanzar este objetivo LFOC emula el comportamiento de la solución óptima de particionado en *cluster*, que hemos aproximado con un simulador en diferentes escenarios. Con este fin, LFOC separa las aplicaciones *streaming* agresoras de las aplicaciones sensibles a la compartición de cache, asignándolas a diferentes particiones.

Hemos implementado LFOC el kernel Linux y evaluamos su efectividad en un sistema real equipado con un procesador Intel Skylake, donde comparamos su eficacia con la de dos estrategias de particionado que optimizan la justicia y el rendimiento global del sistema. Nuestro análisis revela que LFOC es capaz de obtener una mayor reducción de la injusticia mediante un algoritmo con baja sobrecarga que permite su implementación en un sistema operativo real.

Palabras clave— **Procesadores multicore, particionado de caché, clustering, justicia, Intel Cache Allocation Technology, kernel Linux.**

I. Introducción

Los procesadores multicore (CMP) constituyen actualmente la arquitectura dominante en sistemas de computación de propósito general y, probablemente, continuarán siendo un referente en los próximos años. A pesar de las ventajas que los CMPs ofrecen, éstos plantean una serie de retos al software del sistema. Uno de los retos más significativos es la contención que surge al uso de los recursos compartidos [1], que ocurre debido al hecho de que los núcleos no son unidades de procesamiento completamente independientes sino que, típicamente, comparten una caché de último nivel (LLC) y otros recursos de memoria con el resto de cores (cómo el controlador de DRAM, un bus de memoria o una red de interconexión [2], [3]). Las aplicaciones que se ejecutan simultáneamente en los diferentes cores compiten por el uso de estos recursos, lo que puede degradar su rendimiento de forma desigual [4], [3].

El particionado de la LLC compartida (p.ej., asignar una partición disjunta de un tamaño determinado a cada aplicación) ha demostrado ser una técnica efectiva para mitigar los efectos de la contención de recursos compartidos [5], [6], [7], [8], [9], [4]. Recientemente, Intel ha introducido en sus procesadores soporte hardware de particionado (Intel Cache Allocation Technology o CAT) [10], que permite al sistema operativo asignar un cierto número de vías a cada aplicación. Existen múltiples estrategias de gestión de recursos que emplean esta tecnología para optimizar diferentes objetivos como el rendimiento global [8], la justicia [4], o el grado de satisfacción del cliente en sistemas virtualizados [11].

Nuestro trabajo explora cómo emplear Intel CAT a nivel del sistema operativo (SO) para mejorar la justicia del sistema, lo que contribuye a eliminar un número importante de efectos no deseados. Por ejemplo, la contención de recursos compartidos puede causar que el tiempo de ejecución de una aplicación varíe significativamente entre ejecuciones, en función del comportamiento del resto de programas de la carga de trabajo [1], [12]. Además, las aplicaciones con la misma prioridad pueden experimentar una distinta degradación del rendimiento cuando se ejecutan juntas con respecto a su ejecución aislada [13], [2]. Estos problemas hacen que la planificación basada en prioridades no sea efectiva [2], introducen variabilidad en el rendimiento [14] y pueden causar cargos incorrectos en servicios comerciales de computación en la nube [12], donde se cobra a los usuarios por tiempo de CPU. Notablemente, la injusticia también provoca un progreso desigual en los diversos hilos de ejecución de las aplicaciones multihilo HPC (High-Performance Computing) [15], lo que limita la escalabilidad de forma significativa.

Con el objetivo de proporcionar justicia mientras se mantiene un rendimiento global aceptable en el sistema, presentamos LFOC (*Lightweight Fairness-Oriented Cache-clustering*), una estrategia de particionado de cache en cluster a nivel de sistema operativo. Mediante el uso del soporte hardware Intel CAT, LFOC crea dinámicamente un número de particiones de la LLC (*clusters*) de acuerdo a las características de la carga de trabajo, y asigna las aplicaciones a diferentes clusters en función de la contención que generan y de su grado de sensibilidad a compartir la cache con otras aplicaciones.

Las principales contribuciones de nuestro trabajo

[1]Grupo ArTeCS, Dpto. de Arquitectura de Computadores y Automática, Universidad Complutense de Madrid, e-mail: {adriagar,jcsaezal,fcastror,mpmatias}@ucm.es.

son las siguientes:

- Para guiar el diseño de LFOC, aproximamos –empleando un simulador paralelo– la solución de particionado en cluster que optimiza la justicia para diferentes cargas de trabajo. El análisis exhaustivo de la solución óptima revela que la clave para mantener la justicia es identificar las aplicaciones que generan más contención y que son insensibles a la misma (*streaming*). Confinando estas en un espacio reducido de la LLC permite dedicar más memoria libre a las aplicaciones sensibles (*cache-sensitive*).
- A partir de las conclusiones del análisis previo, procedimos a diseñar LFOC, intentando aproximar el comportamiento de la solución óptima. Nuestra estrategia monitoriza continuamente diferentes métricas de cada aplicación en tiempo de ejecución mediante contadores hardware y clasifica las aplicaciones en diferentes categorías basándose su comportamiento en el acceso a la LLC. La información de rendimiento recolectada se usa como entrada a un algoritmo de clustering eficiente. LFOC emplea un mecanismo de baja sobrecarga para aproximar el grado de sensibilidad a compartir la LLC de cada aplicación que ahorra siempre que sea posible costosas operaciones de monitorización (p.ej. medir el rendimiento de una aplicación dinámicamente para diferentes tamaños de cache) usadas por otras estrategias [8].
- Implementamos LFOC en el kernel Linux y lo evaluamos en un sistema real con un procesador Skylake de Intel. Además, comparamos la eficacia de dos políticas previas de particionado de cache –Dunn [4] y KPart[8]– que optimizan la justicia y el rendimiento global, respectivamente. Nuestro análisis revela que LFOC proporciona hasta un 20.5% de mejora de la justicia (9% en media) respecto a Dunn (clustering orientado a justicia), y aporta un mayor grado de rendimiento global y justicia que todas las estrategias analizadas para la mayoría de cargas.

El resto del artículo tiene la siguiente estructura. La sección II presenta conceptos sobre particionado de cache y discute el trabajo relacionado. En la sección III se presenta el análisis de la solución óptima que motiva nuestra propuesta. A continuación, la sección IV expone el diseño de LFOC y destaca sus características principales. En la sección V se realiza la evaluación experimental. Por último, en la sección VI se exponen las conclusiones.

II. Trasfondo y trabajo relacionado

En primer lugar, en esta sección se describen las métricas consideradas para evaluar el grado de justicia y rendimiento global de las diferentes estrategias de particionado. A continuación, se introduce formalmente la noción de particionado y clustering. Finalmente, se discuten algunos problemas asociados y se presenta sobre el trabajo relacionado.

Para medir la degradación del rendimiento de una aplicación en una carga de trabajo hemos utilizado la métrica *Slowdown*, que se define como sigue:

$$Slowdown_{\text{app}} = \frac{CT_{part,app}}{\text{CT}_{alone,app}} \quad (1)$$

donde $CT_{part,app}$ denota el tiempo de ejecución de una aplicación (*app*) cuando comparte el sistema bajo una estrategia de particionado concreta, y $CT_{alone,app}$ es el tiempo de ejecución de la aplicación con el sistema a su completa disposición.

El slowdown de una aplicación se puede definir también como el número medio de instrucciones por ciclo de su ejecución aislada en el sistema con toda la cache disponible ($IPC_{alone,app}$) y respecto a su ejecución con otras aplicaciones bajo una estrategia de particionado determinada ($IPC_{part,app}$):

$$Slowdown_{\text{app}} = IPC_{alone,app}/IPC_{part,app} \quad (2)$$

En trabajos previos sobre justicia en sistemas multicore [2], [4] se define a un algoritmo como justo si las aplicaciones de igual prioridad de la carga de trabajo sufren la misma degradación del rendimiento o *slowdown* por el hecho de compartir el sistema. En este trabajo usamos la métrica *unfairness* para representar esta noción de justicia, que ha sido empleada de forma extensa por múltiples autores [2], [16], [17], [3]. Para una carga de trabajo de n aplicaciones, esta métrica (menor es mejor) se define de la siguiente forma:

$$Unfairness = \frac{MAX(\text{Slowdown}_1,\ldots,\text{Slowdown}_\text{n})}{MIN(\text{Slowdown}_1,\ldots,\text{Slowdown}_\text{n})} \quad (3)$$

Notablemente, de acuerdo a la definición de la métrica *unfairness*, podríamos mejorar su valor simplemente ralentizando ciertas aplicaciones para asegurar cifras similares, pero potencialmente altas, de slowdown. Claramente, esto es inaceptable, ya que podría conseguirse a expensas de una degradación sustancial del rendimiento, lo que nuestra propuesta intenta evitar a toda costa. De esta forma, el valor de la métrica unfairness debe ser reportado junto a el rendimiento global del sistema, cómo hacemos en este artículo. Específicamente, para medir el rendimiento global, empleamos la métrica *System ThroughPut* (STP) [18], [4], –también conocida como Weighted Speedup en [8]–, definida a continuación:

$$STP = \sum_{\text{i}=1}^{\text{n}} \left(\frac{CT_{\text{alone,i}}}{CT_{\text{part,i}}} \right) = \sum_{\text{i}=1}^{\text{n}} \left(\frac{1}{Slowdown_{\text{i}}} \right) \quad (4)$$

Existen principalmente dos estrategias para distribuir el espacio de la LLC entre aplicaciones: *particionado estricto* y *particionado en cluster*.

El *particionado estricto* conlleva asignar una partición distinta con un número determinado de vías a cada aplicación. Sea A una carga de trabajo formada por n aplicaciones $\{a_1, a_2, \cdots, a_n\}$ y sea S un

sistema con una LLC de k vías con $k \geq n$, un particionado estricto de la LLC para A en S se define formalmente como un conjunto $\{w_1, w_2, \cdots, w_n\}$ (con $\sum_{i=1}^{n} w_i = k$) donde w_i denota el número de vías asignadas a la aplicación a_i ($1 \leq w_i \leq k - n + 1$).

En los últimos años se han propuesto diversas estrategias de particionado [5], [6], [7], [19] que persiguen diferentes objetivos de optimización, como la reducción de consumo energético , la mejora del rendimiento global o la justicia del sistema. Estas estrategias emplean algoritmos aproximados para objetivos de optimización específicos. En general, determinar la solución óptima al problema de particionado estricto para un objetivo de optimización determinado se considera un problema NP-duro [6], por lo que determinar la mejor solución haciendo una exploración exhaustiva del enorme espacio de búsqueda resulta inviable. Por ejemplo, para averiguar la solución óptima de particionado estricto para 8 aplicaciones en una plataforma con una LLC de 11 vías, requiere una exploración de 120 soluciones; el número de opciones posibles a considerar en una plataforma con una LLC de 20 vías se incrementa a más de cincuenta mil.

Cuando el número de aplicaciones excede el número de vías disponibles ($n > k$) el particionado estricto de la LLC no es factible; dos o más aplicaciones deben compartir al menos una vía. En trabajos anteriores se ha señalado que aún cuando $n \leq k$, la gruesa granularidad de las particiones (MBs) disponibles en procesadores modernos con la tecnología Intel CAT hacen que el particionado estricto (sin compartir vías entre aplicaciones) sea inapropiado en algunos casos [4], [8]. Debido a la mayor granularidad de la distribución natural de la LLC cuando se comparten vías entre aplicaciones, podríamos obtener en ocasiones un mayor rendimiento (y hasta mejor justicia) si permitimos que se compartan vías entre aplicaciones (*cache clustering*) que por medio de un particionado estricto [19], [8].

Formalmente, definimos un particionado en cluster de la siguiente forma. Sea A una carga de trabajo formada por n aplicaciones $\{a_1, a_2, \cdots, a_n\}$ y sea S un sistema que incorpora una LLC de k vías con $k \geq n$, un particionado en clusters de A en S se define como el conjunto $T = \{C_1, C_2, \cdots, C_m\}$ y el conjunto asociado W de vías asignadas para cada C_i en T, $W = \{w_1, w_2, \cdots, w_m\}$ donde cada C_i es un subconjunto disjunto de A ($C_i \subseteq A$), sujeto a las siguientes restricciones (i) $1 \leq m \leq min(n, k)$, (ii) $C_1 \cup C_2 \cup \cdots \cup C_m = A$, (iii) $\forall i, j, 1 \leq i, j \leq m \wedge j > i, C_i \cap C_j = \emptyset$ y (iv) $(1 \leq w_j \leq k - m + 1) \wedge \sum_{i=1}^{m} w_i = k$. Por simplicidad, nos referiremos a cada elemento en T como un *cluster* o grupo de aplicaciones. De acuerdo a la definición anterior, cada cluster tiene un cierto número de vías de cache asignadas, indicado por el conjunto W (C_1 tiene w_1 vías asignadas), que constituye una de las posibles formas de distribuir el espacio de cache disponible.

Como en el caso del particionado estricto, existen múltiples propuestas de particionado en clusters recientes [11], [8], [4] que persiguen diferentes objetivos de optimización mediante heurísticas que aproximan el óptimo. Notablemente, desde el punto de vista del espacio de búsqueda, encontrar la solución óptima de clustering para un determinado objetivo constituye un problema aún más complejo de resolver que el particionado estricto. Específicamente, para determinar la solución óptima por fuerza bruta, para cada posible clustering del conjunto A (con $min(n, k)$ elementos como mucho), tenemos que determinar la distribución de vías entre clusters para un objetivo de optimización concreto. Cabe destacar que el número posible de soluciones crece exponencialmente con n y k. Por ejemplo, en un sistema con una LLC de vías el número de opciones de clustering diferentes para una carga de trabajo de 8 aplicaciones asciende aproximadamente a 9 millones, mientras que para una carga de 11 aplicaciones existen más de 5000 millones de soluciones posibles.

Existen múltiples trabajos que intentan mitigar el problema de la contención en la LLC con técnicas hardware y software [6], [20], [8], [4], [21], [22]. Estos trabajos emplean estrategias de particionado estricto o en cluster mediante heurísticas que aproximan el óptimo [5], [7], [8], [9]. En un *survey* reciente [6] se discuten las técnicas más efectivas para los diferentes objetivos de optimización. Las particiones de cache se pueden crear en sistemas con soporte hardware específico (Intel CAT) o por medio de soluciones software, en su mayoría basadas en técnicas de coloreado de páginas [23], [24], [25], [26].

Estas técnicas se pueden aplicar en plataformas multicore comerciales [27], aunque están sujetas a ciertas limitaciones, que pueden eliminarse empleando soporte hardware de particionado de cache [4]. Entre las diferentes soluciones hardware, las principales diferencias residen en cómo asignar el número de vías entre las diferentes aplicaciones: hay propuestas basadas en las políticas de reemplazamiento de la cache [28], [29], [30] mientras que otras utilizan el muestreo de los conjuntos y la duplicación de etiquetas de la cache para guiar el particionado [5], [31]

UCP [5] es una estrategia de particionado que intenta mejorar el rendimiento global minimizando el número de fallos en la LLC de cada aplicación. Este algoritmo no intenta determinar la solución óptima sino que emplea un algoritmo de heurístico llamado *lookahead* [5], que utiliza como entrada la tabla de MPKI de cada aplicación (fallos en la LLC por cada mil instrucciones) para todas las asignaciones de vías posibles de la LLC. En la propuesta original, UCP emplea extensiones hardware para construir las tablas en tiempo de ejecución. Desafortunadamente, una década después de la publicación de esta propuesta, estas extensiones hardware aún no se han adoptado en las plataformas comerciales. LFOC usa el algoritmo *lookahead* para distribuir la mayoría del espacio disponible en la LLC entre las aplicaciones *cache-sensitive*, utilizando las tabla de slowdown de cada aplicación como entrada al algoritmo en lugar de las de MPKI; de esta forma se consigue una dis-

tribución más justa del espacio en la cache [32].

Más recientemente, se han propuesto diferentes algoritmos de clustering [8], [4] como alternativa al particionado estricto de cache. KPart [8] es una aproximación de particionado en cluster diseñada para mejorar el rendimiento global; para ellos ejecuta un algoritmo iterativo que crea y combina grupos de aplicaciones mediante clustering jerárquico. Para decidir qué clusters deben combinarse en cada iteración, y cómo distribuir las vías disponibles entre los mismos (particionado entre clusters), la estrategia emplea una métrica de distancia propuesta en [9] así como el algoritmo *lookahead* de UCP [5]. Ambas técnicas se basan en la capacidad de obtener tablas de MPKI e IPC para cada aplicación en tiempo de ejecución. Como se explica en la sección IV, LFOC requiere recopilar una menor cantidad de información que KPart, y evita realizar barridos costosos de vías de cache periódicamente, reduciendo así la sobrecarga de forma significativa.

En [4], Selfa y otros proponen Dunn, una política de particionado diseñada para mejorar la justicia. Esta estrategia agrupa las aplicaciones en clusters aplicando el algoritmo *k-means* [33], que se guía por la fracción de paradas en el pipeline causadas por los fallos en la L2 (en nuestra plataforma esta información se puede obtener con el evento `STALLS_L2_MISS`). Cabe destacar que, de acuerdo con la definición provista en la sección **??**, ésta no es estrictamente una estrategia de clustering, ya que las particiones que realiza se pueden solapar, lo que podría causar interacciones impredecibles entre aplicaciones pertenecientes a diferentes clusters [8].

En nuestra evaluación experimental, comparamos la efectividad de nuestra propuesta (LFOC) con las aproximaciones Dunn y KPart, y demostramos que LFOC consigue generalmente una mayor reducción en la injusticia del sistema. No obstante, queremos subrayar que Dunn y KPart son estrategias de clustering a nivel de usuario, en contraposición a nuestro algoritmo, implementado en el sistema operativo. Las estrategias a nivel de usuario como estas pueden acarrear mayor sobrecarga ya que hacen un uso extensivo de llamadas al sistema para acceder recursos privilegiados que son gestionados directamente por el SO. Por el contrario, LFOC accede a estas interfaces de forma directa utilizando una API de baja sobrecarga a nivel del kernel.

III. Análisis del clustering óptimo

Como se ha mencionado con anterioridad, el diseño de nuestra estrategia se ha inspirado en el comportamiento de la solución óptima de clustering para la justicia. En esta sección se analiza la solución óptima, que podemos aproximar para diferentes cargas de trabajo gracias al simulador PBBCache [32]. Esta herramienta de simulación utiliza datos de rendimiento recolectados offline para diferentes tamaños de cache en una plataforma concreta (p.ej., instrucciones por ciclo, consumo de ancho de banda de memoria, etc.) para determinar el grado de rendimiento global, justicia y otras métricas relevantes para una carga de trabajo bajo un algoritmo de particionado determinado. Una característica clave del simulador PBBCache es su capacidad para determinar las soluciones óptimas de particionado y clustering para diferentes objetivos de optimización (rendimiento global o justicia) mediante la ejecución de un algoritmo de ramificación y poda paralelo. Para aproximar el slowdown de una aplicación, lo cual es necesario para determinar el grado de justicia, PBBCache tiene en cuenta la degradación del rendimiento debido tanto al uso compartido de la cache como a la contención del ancho de banda de memoria (para ello emplea una variante del modelo probabilístico propuesto en [34]).

Para realizar el análisis con el simulador, utilizamos contadores hardware para capturar el valor medio de diferentes métricas de rendimiento durante la ejecución de aplicaciones de las suites SPEC CPU 2006 y 2017 bajo distintos tamaños de cache en un sistema equipado con un procesador Intel Skylake (LLC de 11 vías y 27.5MB, puede encontrarse más información sobre esta plataforma en la sección V). Los valores de estas métricas, que se corresponden con la ejecución de los primeros 1500 billones (EEUU) de instrucciones, se utilizan como entrada del simulador para aproximar la solución óptima de justicia, es decir, la solución a el problema de particionado en clusters que obtiene el valor mínimo de *unfairness* para el máximo rendimiento global (STP) posible.

En los experimentos se emplean cargas de trabajo generadas de forma aleatoria incluyendo un distinto número de aplicaciones SPEC CPU (de 4 a 16). De acuerdo a la información de rendimiento recogida offline se clasifican las aplicaciones en tres clases basándose en el grado de sensibilidad a compartir la cache y la contención que generan: *cache-sensitive*, *light-sharing* y *streaming*. A groso modo, la categoría *cache-sensitive* se aplica a aquellos programas que experimentan caídos significativas del rendimiento cuando se reduce el número de vías de cache para su uso exclusivo; este no es el caso para las aplicaciones *light-sharing* y *streaming*. Los programas *streaming* se caracterizan por exhibir un bajo slowdown para la mayoría de asignaciones de vías y por tener un alto número de fallos en la LLC por ciclo. Las aplicaciones de este tipo son insensibles a compartir la cache y típicamente actúan como programas agresores bajo la perspectiva de las aplicaciones *cache-sensitive* en caso de compartir un cluster, ya que el rendimiento de estas últimas puede degradarse considerablemente. Por el contrario, los programas *light-sharing* no sufren al compartir la LLC ni hacen un uso intensivo de la misma (habitualmente su *working set* cabe en los niveles de caché privados de cada core). En la tabla I se resumen los criterios seguidos para hacer esta clasificación en nuestra plataforma experimental, que están basados en dos métricas: el slowdown de cada aplicación y el número de fallos en la LLC por cada mil instrucciones (LLCMPKC). A modo de ejemplo la figura 1 muestra como el slow-

TABLA I: Clasificación de aplicaciones en función de su comportamiento en el acceso a la cache.

Tipo	Criterio
Streaming	($Slowdown \leq 1.03$ y $LLCMPKC \geq 10$) en al menos una asignación de vías, y $Slowdown < 1.06$ en todas las asignaciones
Sensitive	Si no es *streaming* y $Slowdown \geq 1.05$ para un número de vías ≥ 2
Light-sharing	No es *streaming* ni *sensitive*

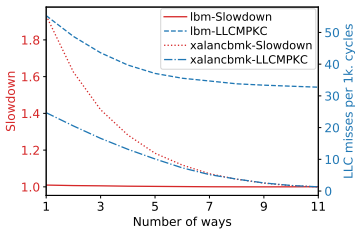


Fig. 1: Slowdown y LLCMPKC para diferentes vías

down y los LLCMPKC varían bajo diferentes asignaciones de vías en el caso de un benchmark *streaming* (`lbm`) y uno *cache-sensitive* (`xalancbmk`).

Tras un meticuloso análisis de las soluciones óptimas de particionado y clustering obtenidas con el simulador para las diversas cargas de trabajo, hemos extraído las siguientes conclusiones:

- La solución que optimiza la justicia aísla todas las aplicaciones streaming en un conjunto reducido de vías (no mayor que 2). En la mayoría de los casos, se usa un cluster de una sola vía para todas las aplicaciones streaming.
- Esta misma solución asigna los programas light-sharing a diferentes clusters con un patrón casi arbitrario. Además, analizando múltiples situaciones se observa que mover de cluster una sola aplicación light-sharing tiene muy poco impacto en el rendimiento y la justicia.
- Como cabía esperar, teniendo en cuenta la definición de la métrica *unfairness*, el número de vías asignadas a las aplicaciones cache-sensitive es crítico tanto para rendimiento global cómo para justicia. Cabe destacar que la métrica *unfairness* refleja el máximo slowdown observado en las aplicaciones de la carga de trabajo, y los benchmarks sensibles típicamente sufren una importante degradación del rendimiento si no se satisfacen sus requisitos de espacio en la cache.
- El beneficio de asignar programas a particiones separadas (incluso de forma óptima) decrece drásticamente cuando el número de aplicaciones es cercano al número de vías. Cuando el número de aplicaciones es igual al de vías, cada aplicación sólo puede recibir una única vía siguiendo un particionado estricto –ésta es la única opción factible– lo que genera un alto *unfairness*. La conclusión que sacamos es que las políticas de clustering son claramente superiores a las de particionado a medida que crece el número de vías con respecto al de aplicaciones.

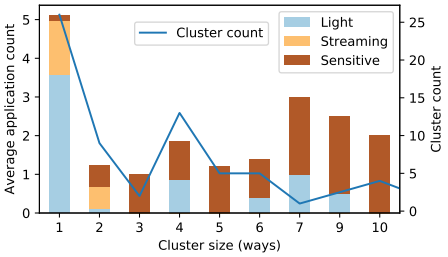


Fig. 2: Número de clusters y desglose de las aplicaciones en categorías para diferentes tamaños de cluster.

Para profundizar en el comportamiento general de la solución óptima, la figura 2 muestra el número medio de aplicaciones por tamaño de cluster, además del número total de clusters –agrupados por su tamaño en vías– que la solución construye para un subconjunto de las cargas de trabajo exploradas: 20 mezclas aleatorias formadas cada una por 10 aplicaciones. Los datos mostrados en la figura confirman las tres observaciones mencionadas. Primero, las aplicaciones streaming son relegadas a clusters con una sola vía. En términos relativos, más del 87% de las aplicaciones streaming son asignadas a este tipo de clusters, mientras que el resto se asignan a clusters de dos vías. Segundo, las aplicaciones light-sharing se asignan a clusters con tamaños muy dispares; sin embargo, la mayoría de estos programas pertenecen a clusters de una sola vía. Tercero, los resultados revelan que las aplicaciones cache-sensitive están presentes de forma predominante en clusters de gran tamaño. Específicamente, más del 77% de las aplicaciones sensibles se encuentran en clusters de 4 o más vías. Finalmente, cabe destacar que los clusters de una sola vía con alto número de aplicaciones aparecen con frecuencia en las soluciones óptimas para las diferentes cargas de trabajo.

IV. Diseño

En esta sección se describe el funcionamiento de nuestro algoritmo de clustering a alto nivel y cómo se clasifican las aplicaciones dinámicamente con información proporcionada por los contadores hardware.

A. Funcionamiento del algoritmo

Nuestra estrategia LFOC se ha implementado en el kernel Linux como una extensión del planificador del SO. Específicamente, se ha incorporado en un módulo cargable del kernel como un *plugin* de monitorización de la herramienta PMCTrack [17]. Esta herramienta proporciona una API a nivel del kernel para recursos de acceso privilegiado como los contadores hardware (Performance Monitoring Counters o PMCs) o la interfaz con Intel CAT (p.ej., soporte para hacer particionado de la cache por vías).

LFOC clasifica dinámicamente las aplicaciones en tres clases basándose en su comportamiento de cache –*light-sharing*, *streaming* y *sensitive*– y asigna cada aplicación a una partición cuyo tamaño se determina en función de sus propiedades.

Cuando una aplicación comienza su ejecución su comportamiento en la caché se desconoce. Por esta

razón, se le asigna una clase especial (*unknown*) que indica que deberá realizar un período de calentamiento (3 intervalos de monitorización). Cualquier información recolectada durante este período no se emplea para clasificar las aplicaciones, con el fin de evitar malas predicciones asociadas con la alta variabilidad en los fallos de cache típicamente presente durante el inicio de la ejecución.

Periódicamente, nuestra extensión del planificador activa el algoritmo de particionado basado en las conclusiones del análisis de la sección III. A grandes rasgos, el algoritmo reserva dos vías de cache para asignar los programas streaming y el resto de vías se distribuyen entre las aplicaciones sensibles, que se encuentran en diferentes particiones de cache. El tamaño de estas particiones se determina a través del algoritmo lookahead [5], utilizando como entrada la curva de slowdown de cada aplicación (p.ej., slowdown medido online para diferentes vías de cache) construida utilizando los valores de IPC medidos en tiempo de ejecución. Con esta distribución de vías para las aplicaciones cache-sensitive, LFOC intenta satisfacer los requisitos de cache basándose en sus grados de sensibilidad a compartirla. Finalmente, las aplicaciones light-sharing se distribuyen entre las diferentes particiones priorizando a aquellas con aplicaciones streaming, siguiendo la estrategia que típicamente utiliza la solución óptima.

B. Clasificación de aplicaciones

Una vez se finaliza el período de calentamiento, LFOC inicia su modo de muestreo cuyo objetivo es determinar la clase a la que pertenece cada aplicación basándose en su sensibilidad a compartir la LLC y su uso de memoria. Esto es crucial para decidir cómo particionar la LLC entre las aplicaciones de una carga de trabajo y cuáles son compatible para, en caso de ser necesario, compartir una partición [6].

El modo de muestreo está inspirado en la técnica propuesta en [8], que se describe a continuación. Se crean dos particiones disjuntas de la cache que cubren todo el espacio de la misma; la primera, denominada *partición de muestreo*, se reserva para que la aplicación que activó el modo de muestreo y la segunda se dedica al resto de aplicaciones. Para determinar la clase de la aplicación –basándonos en los criterios presentados en la sección III– el valor de los diferentes eventos obtenidos con contadores hardware (p.ej., número de instrucciones retiradas, ciclos y fallos en la LLC) a medida que aumentamos el tamaño de la partición de muestreo. Nótese que para las aplicaciones sensibles también se construye la curva de slowdown, necesaria para decidir el tamaño de sus particiones. Una vez finaliza el proceso de muestreo, LFOC retorna al modo normal de operación descrito anteriormente.

En la propuesta original [8], el tamaño de la primera partición varía desde 1 al número de vías menos 1, mientras que el tamaño de la segunda partición (complementaria) decrece proporcionalmente. Este barrido completo es necesario en la versión dinámica de KPart, que se basa en la obtención de los valores de IPC y LLCMPKI para cada número de vías y aplicación de la carga de trabajo. Hemos observado que esta estrategia puede causar sobrecargas importantes debido a que la asignación de cache durante el modo de muestreo es típicamente subóptima. La aplicación a muestrear recibe cada vez un mayor espacio, mientras que el resto de aplicaciones disponen de un cluster cada vez más pequeño. Esto provoca con frecuencia una importante degradación del rendimiento global y la justicia, especialmente cuando existen múltiples aplicaciones sensibles y streaming en la carga de trabajo.

Para solventar estas limitaciones, LFOC detiene inmediatamente el proceso de muestreo si al incrementar el tamaño de la partición no se observa un cambio significativo en la información obtenida para el algoritmo de clustering. En primer lugar, cuando el ratio de fallos de LLC cae por debajo de un cierto *umbral inferior*, el rendimiento no va a mejorar aunque se incremente el espacio de cache dedicado a la aplicación, por lo que se esperan unos valores de IPC –usados para construir las tablas de slowdown– muy similares a partir de ese punto. En segundo lugar, las aplicaciones streaming exhiben habitualmente un incremento muy bajo del rendimiento cuando se les concede más espacio de cache. En estos escenarios, LFOC interrumpe el proceso de muestreo y determina la clase de la aplicación. En la práctica, para identificar con éxito muchas aplicaciones streaming y light-sharing –cuyas curvas de slowdown no se requieren para el algoritmo de clustering– solo se necesita muestrear un reducido número de vías. Cuando una aplicación es sensible y el proceso de muestreo se cancela debido al primer criterio, LFOC utiliza la última muestra de IPC recogida para aproximar el rendimiento con tamaños superiores, lo que es necesario para construir la tabla de slowdown completa.

Debido a que una aplicación puede atravesar varias fases de ejecución, la clasificación puede ser incorrecta a lo largo del tiempo. Si se determina incorrectamente la clase de una aplicación se pueden realizar particionados subóptimos en ciertos intervalos y, por tanto, degradar el rendimiento global y la justicia. Activar periódicamente el modo de muestreo ayuda a mitigar este problema aunque, desafortunadamente, tiene la contrapartida de introducir una sobrecarga extra significativa. Para evitar esto, el SO monitoriza constantemente el valor de LLCMPKC de cada aplicación y la fracción de paradas del pipeline incurridas por fallos de cache de alta latencia (aproximados mediante el evento `STALLS_L2_MISS`, también usado en [4]), y utiliza varias heurísticas para capturar los cambios de clase. En concreto, se detecta un cambio de clase de una aplicación light-sharing si entra una fase intensiva en memoria, es decir, si la media de LLCMPKC medida durante los últimos cinco intervalos de monitorización exceden el umbral `high_threshold` (10 en nuestra plataforma experimental, como se muestra en la tabla I para las apli-

caciones streaming) o si la fracción media de stalls causados por fallos de alta latencia es mayor del 25%. Esta estrategia consigue filtrar los picos en las métricas mencionadas y permite identificar de forma efectiva las fases intensivas en memoria. En cambio, para los programas streaming, que LFOC asigna en su mayoría a clusters de una vía, el modo de muestreo se activa si la media de LLCMPKC cae por debajo de un umbral `low_threshold` (definido como 30% del `high_threshold`). Finalmente, para las aplicaciones sensibles, LFOC asocia un *tamaño crítico*, definido como el número de vías dónde el slowdown cae por debajo de 1.05. El tamaño crítico se determina durante el último intervalo de muestreo de la aplicación y se registra un cambio de clase cuando entra en una fase no intensiva en memoria (criterio opuesto al de las aplicaciones light-sharing)[1] con un uso de cache inferior al *tamaño crítico*, o cuando la media de LLCMPKC es mayor que el umbral `high_threshold` para un tamaño de cache mayor que el valor crítico.

V. Experimentos

Para evaluar la eficacia de nuestra propuesta de particionado en cluster a nivel de SO, procedimos a su implementación en el kernel Linux (v4.9.160). Para la evaluación experimental empleamos un servidor que incorpora un procesador multicore Xeon Gold 6138 "Skylake" (2Ghz). Este procesador integra una LLC (L3) de 27.5MB y 11 vías, con soporte para particionado; cada core incluye una cache L1 de 1.25MB y una L2 de 20MB, ambas privadas.

En esta plataforma realizamos una comparativa experimental de LFOC con respecto al kernel Linux por defecto (*stock linux*)–sin particionar la LLC– , y con las políticas de particionado Dunn [4] y KPart [8], específicamente diseñadas para optimizar la justicia y el rendimiento global respectivamente. Para realizar una comparación justa, hemos seguido una metodología similar a la descrita en los correspondientes artículos [4], [8]. Esencialmente, realizamos experimentos con cargas de trabajo que incluyen aplicaciones secuenciales de las suites SPEC CPU 2006 y 2017, y ejecutamos cada aplicación durante un número fijo de instrucciones (150 billones americanos). Específicamente, garantizamos que todas las aplicaciones de las cargas son ejecutadas simultáneamente, y que cuando una termina sus instrucciones correspondientes, el programa se reinicia reiteradamente hasta que la aplicación más larga del conjunto finaliza tres ocasiones. Entonces calculamos el *unfairness* y *STP* (rendimiento global), utilizando la media geométrica de los tiempos de ejecución de cada programa.

La figura 3 muestra la composición de las cargas usadas en nuestros experimentos, formadas por benchmarks de las suites SPEC CPU 2006 y 2017. Cabe destacar que se han seleccionado aplicaciones de ambas suites para contar con un amplio rango de programas *streaming* y *cache-sensitive*, ya que la mayoría de aplicaciones en ambas suites exhiben un comportamiento *light-sharing* e insensible a compartir cache en nuestra plataforma. Esto es debido en parte a la granularidad gruesa de las particiones de cache que se pueden crear en este sistema: tamaño mínimo de 2.5MB. Como se puede observar, hemos considerado cargas de trabajo de 8, 12 y 16 aplicaciones con el fin de analizar el impacto que el tamaño de la carga tiene en las mejoras del grado de justicia en el sistema que consigue cada estrategia de particionado.

A. Evaluación de los algoritmos de clustering

Nuestra meta es medir el grado de justicia y rendimiento global de cada estrategia de clustering independientemente de las sobrecargas asociadas debidas a la ejecución del algoritmo, monitorización de rendimiento o latencia del particionado de la cache.

Para evaluar la efectividad de cada algoritmo, consideramos cargas de trabajo con aplicaciones con un claro comportamiento de una clase para la mayoría de su ejecución. Además, implementamos los algoritmos de clustering utilizado por KPart, Dunn y LFOC en el simulador descrito en la sección III, el cual acepta como entrada el valor medio de las diferentes métricas de rendimiento recogidas offline para diferentes tamaños de cache. Para realizar los experimentos bajo un algoritmo concreto, ejecutamos el simulador con anterioridad a cada carga de trabajo para establecer las particiones de cache y a cuál pertenece cada aplicación. Tras esto, se establecen las particiones concretas por cada proceso desde espacio de usuario, utilizando la herramienta PMC-Track [17], y procedemos a lanzar la carga de trabajo, que usará el mismo particionado a lo largo de la ejecución. Para realizar una comparación exhaustiva, también mostramos los resultados de una política ideal de particionado, referida como Best-Static, la cual establece las particiones de cache y la asignación de aplicaciones basándose en la solución óptima de justicia determinada por el simulador.

La figura 4 muestra el grado de injusticia y rendimiento global provisto por las diferentes estrategias; los valores han sido normalizados con respecto a Stock-Linux (sin particionado). Los resultados revelan que la aproximación Dunn, diseñada para optimizar justicia, exhibe un comportamiento variable para el conjunto de cargas de trabajo; en algunas es capaz de reducir el *unfairness* hasta un 15.5%, pero para otras causa una sustancial degradación de la justicia (hasta un factor de hasta 1.14x) relativo a Stock-Linux. Encontramos que esto se debe a utilizar exclusivamente la métrica `STALLS_L2_MISS`; cuanto más alto sea el valor de este evento, mayor es el número de vías de cache asignadas a la aplicación por Dunn [4]. En específico, observamos que algunas aplicaciones agresoras streaming y cache-insensitive, como `GemsFDTD` o `fotonik3d`, exhiben valores altos de este evento, ya que su rendimiento se ve altamente afectado por los accesos accesos a memoria. Estas aplicaciones podrían ser asignadas a la misma

[1] La cantidad de cache usada por una aplicación se recoge mediante la Intel Cache Monitoring Technology.

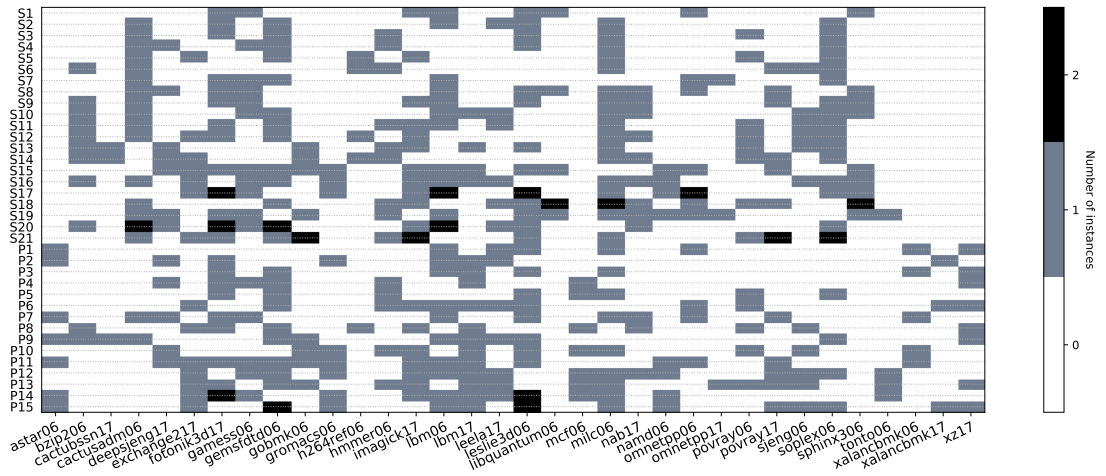


Fig. 3: Cargas de trabajo de nuestros experimentos. Cada celda indica el número de instancias de un benchmark (eje x) en una carga de trabajo (eje y).

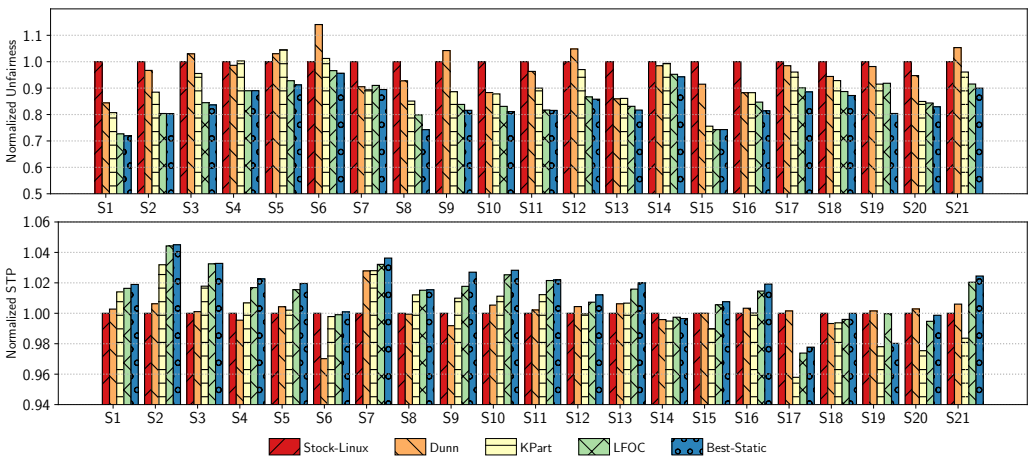


Fig. 4: Reducciones medias de injusticia e incrementos de rendimiento de la versión estática de los algoritmos de clustering.

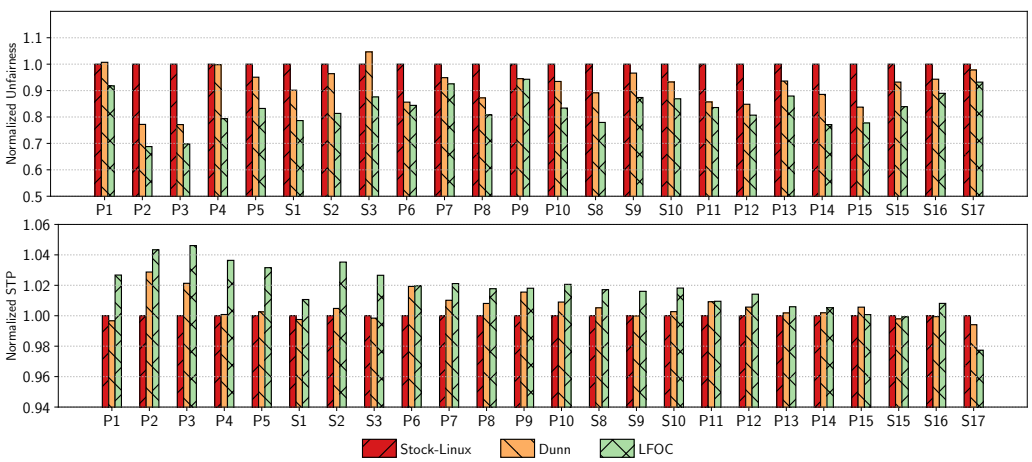


Fig. 5: Reducciones medias de injusticia e incrementos de rendimiento de la versión dinámica de los algoritmos de clustering.

(o solapadas) partición de cache que otras sensibles, como `soplex` o `omnetpp`, las cuales experimentan valores similares de este evento. Esto podría causar una importante degradación del rendimiento global y la justicia. Basándonos en esta observación, concluimos que utilizar únicamente el evento `STALLS_L2_MISS` no es suficiente para dirigir las políticas de particionado.

También observamos que el algoritmo de clustering de KPart, diseñado para optimizar el rendimiento global, aporta un incremento moderado del mismo[2] en las cargas de trabajo exploradas (hasta un 3%). No obstante, esta aproximación sí que aporta en sustanciales reducciones en la injusticia (8.6% en media).Aunque, hemos observado que la estrategia más sencilla y de menor sobrecarga de LFOC provee una mayor justicia que KPart para la mayoría de car-

[2]En el artículo original se reporta un incremento medio del 24% en el rendimiento global en otra plataforma, en el que la composición de las cargas no está especificada.

gas de trabajo (hasta un 27.3%, y 14% en media relación a Stock Linux). Al mismo tiempo, LFOC consigue un mayor rendimiento global en general, y se comporta en un rango similar (1.8% en media) a la versión *Best Static* (política de particionado óptima) en estos escenarios de cargas de trabajo.

B. Estudio de las estrategias dinámicas

En los experimentos de esta sección empleamos nuestra la a nivel del kernel de LFOC y creamos una implementación a nivel de usuario de Dunn tal y cómo se propuso originalmente en [4]. Una ventaja de esta estrategia es que sólo requiere la monitorización del evento `STALLS_L2_MISS`, esta simplicidad es destacable con respecto a la complejidad de KPart, el cual depende de la capacidad para recoger de forma precisa una cantidad sustancial de información de rendimiento online para cada aplicación (los valores de LLCMPKI e IPC para cada número de vías) para aplicar el algoritmo de clustering.

TABLA II: Ejecución en ms de los algoritmos KPart y LFOC

#Apps.	4	5	6	7	8	9	10	11
LFOC	0.00151	0.00154	0.00163	0.00174	0.00174	0.00182	0.00191	0.00216
KPart	0.51800	0.79600	1.21800	1.48100	2.01200	2.74200	3.32000	4.14000

En un intento de evaluar la versión dinámica de KPart –referido como KPart-Dynaway [8]– consideramos la implementación a nivel de usuario creada para los autores [35]. Desafortunadamente, esta implementación, que consiste aproximadamente de 4 mil líneas de código C++ y que hace un uso intensivo de la librería de álgebra lineal Armadillo, fue específicamente diseñada para la plataforma hardware en la que los autores realizaron los experimentos [8], y hace una serie de asunciones que no son trasladables a nuestro entorno experimental(p.ej., el número de vías de cache no puede ser inferior al número de aplicaciones de la carga de trabajo). Debido a estas limitaciones específicas a la plataforma, la ejecución de KPart-Dynaway falla tras la primera ejecución del algoritmo de particionado, impidiendo la ejecución de las cargas de trabajo consideradas. No obstante, para subrayar las enormes diferencias entre el complejo algoritmo de particionado de KPart y el usado por nuestra aproximación para diferente número de aplicaciones, la tabla II muestra el tiempo de ejecución de ambos algoritmos (compilados con optimización agresiva). En esta evaluación, fuimos capaces de recopilar la información de la implementación de KPart instrumentando el código del algoritmo de particionado que se completa con éxito (para tamaños de carga inferiores a 11 aplicaciones) justo antes de que se produzca un fallo de ejecución. Tal y como se puede observar, el tiempo de ejecución de LFOC (2μs) es hasta 3 ordenes de magnitud inferiores que el de KPart, el cual puede tardar hasta 4ms en completarse para 11 aplicaciones (un tiempo algo superior al tick de planificación del kernel Linux). Como hemos mostrado en la sección anterior, la mayor complejidad de la implementación de KPart no se traduce en una mejor de la justicia con respecto a nuestra propuesta de baja sobrecarga.

En la implementación a nivel del sistema operativo de LFOC, se recoge información de los contadores hardware cada 100M de instrucciones en el modo de operación normal (ver sección IV) y cada 10M de instrucciones durante el modo de muestreo. Al usar una venta de instrucciones más corta se permite reducir el tiempo requerido para realizar un barrido completo de las vías para construir las curvas. El tiempo requerido por el modo de muestreo es de 6.2ms en media. Notablemente, observamos que en la mayoría de casos no es necesario realizar un barrido completo ya que LFOC no requiere métricas detalladas de cada vía para todas las aplicaciones, a diferencia de la aproximación de KPart. En nuestros experimentos, el algoritmo de particionado para Dunn y LFOC se ejecuta cada 500ms, ya que es la configuración originalmente en Dunn [4].

En la figura 5 se muestran los valores de injusticia y rendimiento global de las versiones dinámicas de Dunn y LFOC para las diferentes cargas de trabajo. Nótese que consideramos mezclas de programas adicionales (Pi workloads) que incluyen aplicaciones como `xz`, `astar`, `mcf` or `xalancbmk`, las cuales exhiben prolongadas fases de ejecución con un comportamiento con diferentes grados de intensidad en memoria. Algunas de estas aplicaciones atraviesan diferentes fases de sensibilidad a compartir la LLC, por lo que la ejecución si particionado (Stock-Linux) experimenta unos altos valores de *unfairness* en estos escenarios. Esa es la razón por la que Dunn exhibe un mejor comportamiento relativo a los experimentos estáticos. Aun así, LFOC es capaz de aportar un mayor rendimiento global que Dunn, y mejora la justicia de forma generalizada (hasta 20.5% para P4, y 9% en media). Con respecto a Stock-Linux, LFOC reduce la injusticia en un 16.7% en media.

VI. Conclusiones

En este artículo se ha presentado LFOC, una estrategia de particionado en clusters a nivel del sistema operativo que emplea el soporte para particionado de cache Intel-CAT, que permite mejorar la justicia en los procesadores multicore comerciales mientras se mantiene un rendimiento global aceptable. LFOC clasifica las aplicaciones en tres clases en función su grado de intensidad en memoria y sensibilidad a compartir la LLC, y asegura que los programas agresores streaming se aíslan en particiones de cache pequeñas de forma que no perjudiquen a las aplicaciones sensibles, que reciben un tamaño de partición acorde a sus necesidades. De esta forma, LFOC intenta emular el comportamiento de la solución óptima de particionado en clusters, que aproximamos mediante un simulador. Además, implementamos LFOC en el kernel Linux y evaluamos su efectividad en una plataforma multicore comercial con un procesador Intel Skylake. Nuestros experimentos revelan que LFOC es capaz de proporcionar un incremento de la justicia en media de

16.7% en relación a stock Linux. A la vez, se ha demostrado que LFOC funciona mejor que las propuestas existentes de clustering, una de las cuales fue específicamente diseñada para mantener la justicia [4]

Uno de los aspectos fundamentales de LFOC es un algoritmo de clustering de baja sobrecarga, las heurísticas que emplea en tiempo de ejecución para clasificar las aplicaciones, y su habilidad para repartir de forma justa el espacio de la LLC entre aplicaciones empleando un conjunto de métricas reducido, que puede ser obtenido en tiempo de ejecución sin tener que hacer siempre un barrido completo de vías para todas las aplicaciones, a diferencia de otros algoritmos de particionado en clusters [8].

Agradecimientos

Este trabajo ha sido financiado por la Unión Europea (FEDER), el MINECO y la Comunidad de Madrid, bajo los proyectos TIN 2015-65277-R y S2018/TCS-4423. El trabajo de Adrian Garcia-Garcia está financiado por una beca FPU UCM.

Referencias

[1] Sergey Zhuravlev et al., "Survey of scheduling techniques for addressing shared resources in multicore processors," *ACM Comput. Surv.*, vol. 45, pp. 4:1–4:28, Dec. 2012.

[2] Eiman Ebrahimi et al., "Fairness via source throttling: a configurable and high-performance fairness substrate for multi-core memory systems," in *15th Int'l Conf. Architectural Support Programming Lang. and Oper. Syst. (ASPLOS 10)*, 2010, pp. 335–346.

[3] A. Garcia-Garcia, J. C. Saez, and M. Prieto-Matias, "Contention-aware fair scheduling for asymmetric single-ISA multicore systems," *IEEE Transactions on Computers*, vol. 67, no. 12, pp. 1703–1719, Dec 2018.

[4] V. Selfa et al., "Application clustering policies to address system fairness with intel's cache allocation technology," in *2017 26th International Conference on Parallel Architectures and Compilation Techniques (PACT)*.

[5] Moinuddin K. Qureshi and Yale N. Patt, "Utility-based cache partitioning: A low-overhead, high-performance, runtime mechanism to partition shared caches," in *Proceedings of MICRO 06*, 2006, pp. 423–432.

[6] Sparsh Mittal, "A survey of techniques for cache partitioning in multicore processors," *ACM Comput. Surv.*, vol. 50, no. 2, pp. 27:1–27:39, May 2017.

[7] Chenjie Yu and Peter Petrov, "Off-chip memory bandwidth minimization through cache partitioning for multi-core platforms," in *Proceedings of the 47th Design Automation Conference*, 2010, DAC '10, pp. 132–137.

[8] N. El-Sayed et al., "Kpart: A hybrid cache partitioning-sharing technique for commodity multicores," in *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, Feb 2018, pp. 104–117.

[9] Anurag Mukkara, Nathan Beckmann, and Daniel Sanchez, "Whirlpool: Improving dynamic cache management with static data classification," in *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems*, 2016, ASPLOS '16, pp. 113–127.

[10] K. Nguyen, "Introduction to cache allocation technology in the intel xeon processor e5 v4 family," `https://software.intel.com/en-us/articles/introduction-to-cache-allocation-technology`, 2016.

[11] Liran Funaro, Orna Agmon Ben-Yehuda, and Assaf Schuster, "Ginseng: Market-driven llc allocation," in *Proceedings of the 2016 USENIX Annual Technical Conference*, 2016, USENIX ATC '16, pp. 295–308.

[12] J. Feliu et al., "Perf & fair: a progress-aware scheduler to enhance performance and fairness in SMT multicores," *IEEE Transactions on Computers*, vol. PP, no. 99, 2016.

[13] Onur Mutlu and Thomas Moscibroda, "Stall-time fair memory access scheduling for chip multiprocessors," in *40th Ann. IEEE/ACM Int'l Symp. on Microarchitecture (MICRO 07)*, 2007, pp. 146–160.

[14] H. Yun et al., "Memory bandwidth management for efficient performance isolation in multi-core platforms," *IEEE Transactions on Computers*, Feb 2016.

[15] K. Van Craeynest et al., "Fairness-aware scheduling on single-ISA heterogeneous multi-cores," in *22nd Conf. Parallel Arch. Compilation Techniques (PACT 13)*.

[16] Di Xu et al., "Providing fairness on shared-memory multiprocessors via process scheduling," in *Proc. ACM Int'l Conf. Measurement and Modeling Comp. Syst. (SIGMETRICS 12)*, 2012, pp. 295–306.

[17] Juan Carlos Saez et al., "PMCTrack: Delivering performance monitoring counter support to the OS scheduler," *The Computer Journal*, vol. 60, no. 1, pp. 60–85, 2017.

[18] S. Eyerman et al., "System-level performance metrics for multiprogram workloads," *IEEE Micro*, May 2008.

[19] Jacob Brock et al., "Optimal cache partition-sharing," in *Proceedings of the 2015 44th International Conference on Parallel Processing (ICPP)*, 2015, ICPP '15.

[20] David Lo et al., "Heracles: improving resource efficiency at scale," in *Proceedings of the 42nd Annual International Symposium on Computer Architecture*, 2015.

[21] Haishan Zhu and Mattan Erez, "Dirigent: Enforcing qos for latency-critical tasks on shared multicore systems," in *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems*, 2016, ASPLOS '16.

[22] Juan Carlos Sáez et al., "Improving priority enforcement via non-work-conserving scheduling," in *ICPP '08: Proceedings of the 2008 37th International Conference on Parallel Processing*, 2008, pp. 99–106.

[23] Timothy Sherwood et al., "Reducing cache misses using hardware and software page placement," in *Proceedings of the 13th International Conference on Supercomputing*, 1999, ICS '99, pp. 155–164.

[24] Ying Ye et al., "Coloris: A dynamic cache partitioning system using page coloring," in *Proceedings of the 23rd International Conference on Parallel Architectures and Compilation*, 2014, PACT '14, pp. 381–392.

[25] Xiao Zhang et al., "Towards practical page coloring-based multicore cache management," in *Proceedings of the 4th ACM European Conference on Computer Systems*, 2009, EuroSys '09, pp. 89–102.

[26] Alberto Scolari et al., "A software cache partitioning system for hash-based caches," *ACM Trans. Archit. Code Optim.*, vol. 13, no. 4, pp. 57:1–57:24, Dec. 2016.

[27] Heechul Yun et al., "PALLOC: DRAM bank-aware memory allocator for performance isolation on multicore platforms," in *20th Real-Time Embedded Tech. and Applications Symp. (RTAS 14)*, 2014, pp. 155–166.

[28] R Manikantan, Kaushik Rajan, and R Govindarajan, "Probabilistic shared cache management (prism)," in *Proceedings of the 39th Annual International Symposium on Computer Architecture*, 2012, ISCA '12, pp. 428–439.

[29] Samira Manabi Khan et al., "Improving cache performance using read-write partitioning," in *20th IEEE International Symposium on High Performance Computer Architecture, HPCA 2014*, 2014, pp. 452–463.

[30] Ruisheng Wang and Lizhong Chen, "Futility scaling: High-associativity cache partitioning," in *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*, 2014, MICRO-47, pp. 356–367.

[31] Lavanya Subramanian et al., "The application slowdown model: Quantifying and controlling the impact of inter-application interference at shared caches and main memory," in *Proceedings of the 48th International Symposium on Microarchitecture*, 2015, MICRO-48, pp. 62–75.

[32] Adrian Garcia-Garcia et al., "PBBCache: A parallel branch-and-bound based cache-partitioning simulator," *Submitted for review to International Journal of Computatioinal Science*, 2019.

[33] JA Hartigan et al., "Algorithm AS 136: A K-means clustering algorithm," *Applied Statistics*, 1979.

[34] Tomer Y. Morad et al., "Efs: Energy-friendly scheduler for memory bandwidth constrained systems," *Journal of Parallel and Distributed Computing*, 2016.

[35] N. El-Sayed et al., "Source code of kpart," https://github.com/Nosayba/kpart, 2018.

Tasa de aciertos ideal y predecible para la transposición de matrices en caches de datos

Alba Pedro-Zapater[1], Clemente Rodríguez[2], Juan Segarra[3], Rubén Gran Tejero[4] y Víctor Viñals-Yúfera[5]

Resumen— **La transposición de matrices es una operación fundamental pero que ofrece una tasa de aciertos en datos muy baja para matrices grandes. Además esta tasa de aciertos no se puede predecir fácilmente, lo cual es un gran inconveniente para el análisis de sistemas de tiempo real. En este trabajo estudiamos el problema de la transposición de matrices y analizamos su tasa de aciertos en datos para una implementación *tiling*, asumiendo una cache de datos LRU. Obtenemos expresiones teóricas que garantizan la tasa de aciertos ideal en datos dependiendo de los parámetros de cache y el tamaño de *tile*. Con estas expresiones se pueden ajustar fácilmente el tamaño de *tile* y la configuración de cache de manera que la tasa de aciertos sea óptima y predecible. Comparamos nuestros resultados con los del algoritmo *cache oblivious* de la transposición de matrices demostrando que, con el tamaño de *tile* adecuado, la versión *tiling* da lugar a tasas de aciertos en datos iguales o mejores que en *cache oblivious*. También analizamos el tiempo de ejecución de la transposición de matrices en hardware real. Nuestros resultados muestran que la política pseudo-LRU tiene un comportamiento muy similar a LRU y también podemos apreciar otros factores como la prebúsqueda de datos.**

Palabras clave— **tiling transposición matrices caches**

I. Introducción

La transposición de matrices es una operación fundamental en álgebra lineal, transformaciones de fourier, etc. y tiene muchas aplicaciones en áreas como el análisis numérico, el procesamiento de imágenes y gráficos. Aunque la transposición es un problema muy simple, su implementación básica presenta una tasa de aciertos muy bajo para matrices de grandes dimensiones [1]. Esto se debe a que el acceso a elementos consecutivos en la matriz (los cuales probablemente encajarán en la misma línea de cache y por lo tanto presentarán un reuso temporal) sufren de muchos accesos a datos entre ellos. Para superar este problema pueden aplicarse transformaciones de código conocidas como *Tiling* (o *blocking*) presente en todas las bibliotecas de alto rendimiento. Esta transformación se divide todo el problema en pequeños bloques (*tiles*) que encajan en la cache [2]. Por consiguiente, los *tiles* de la matriz original se transponen de manera secuencial, con lo cual cada *tile* permanece cacheado mientras está siendo procesado, evitando así los fallos por capacidad. Esta transformación implica añadir bucles externos al código original, así que la secuencia global de accesos no se expande si no que se mantiene localizada en los *tiles* procesados. Aunque el *tiling* incremente de manera efectiva la tasa de aciertos de la cache de datos, tiene dos importantes desventajas. La primera, añadir bucles implica que se necesitan más instrucciones para llevar a cabo la transposición de la matriz, con su correspondiente tiempo de ejecución La segunda, que el tamaño de la matriz a transponer y la configuración específica de la cache en el sistema (número de conjuntos y vías, tamaño de línea de cache, política de reemplazo, prebúsqueda, etc.) afectarán a la tasa de aciertos del código con la transformación de *tiling*. Estos inconvenientes son importantes en la computación en la nube (donde probablemente la configuración de la cache no es conocida para el usuario y puede cambiar) y en sistemas de tiempo real críticos (donde el cálculo del peor caso de tiempo de ejecución (*WCET*) necesita predictibilidad) [3].

Por otro lado, podemos usar un algoritmo *cache oblivious*. Básicamente estos algoritmos son versiones recursivas de los algoritmos *tiling*, así que el rendimiento óptimo de la cache se alcanza por la propia naturaleza de la recursividad. Esto es, un algoritmo *cache oblivious* no necesita ningún parámetro basado en un conocimiento explícito de la configuración de la cache. Centrándonos en la transposición de matrices, esto significa que no requiere del parámetro de tamaño de *tile* (obligatorio en *tiling*) ya que cada iteración de la recursividad divide la matriz a transponer en muchas matrices más pequeñas hasta que alcanzar un tamaño de 2×2 elementos. Como Tsifakis et al. concluyen [4], "predecir a priori como el algoritmo *oblivious* va a comportarse no es trivial", así que los inconvenientes anteriores siguen presentes.

En este trabajo analizaremos la transposición de matrices desde una perspectiva teórica, y como los parámetros de cache (número de conjuntos, vías, y tamaño de línea) en una cache LRU afectan a la tasa de aciertos en datos en la versión *tiling* del algoritmo. Validamos nuestro análisis por medio de simulaciones considerando un amplio rango de parámetros. También comparamos nuestros resultados con los de una implementación *cache oblivious*. Finalmente analizamos el rendimiento de la transposición de matrices en hardware real (con su configuración específica de cache) para verificar que cumple con nuestras conclusiones teóricas. Nuestras contribuciones pueden reco-

[1]Dpt. Informática e Ing. de Sist. I3A, Universidad de Zaragoza, Spain. HiPEAC, e-mail: albapz@unizar.es.
[2]Dpt. Arquitectura y Tecnología de Computadores, Universidad del País Vasco, Spain. HiPEAC, e-mail: acprolac@ehu.es.
[3]Dpt. Informática e Ing. de Sist. I3A, Universidad de Zaragoza, Spain. HiPEAC, e-mail: jsegarra@unizar.es.
[4]Dpt. Informática e Ing. de Sist. I3A, Universidad de Zaragoza, Spain. HiPEAC, e-mail: rgran@unizar.es.
[5]Dpt. Informática e Ing. de Sist. I3A, Universidad de Zaragoza, Spain. HiPEAC, e-mail: victor@unizar.es.

gerse en:

- Expresiones teóricas para estimar el comportamiento ideal de una cache de datos (fallos obligatorios) en la transposición de matrices, independientes de la cache y el algoritmo de transposición
- Expresiones teóricas para una configuración óptima de una cache de datos LRU para la versión *tiling* de la transposición de matrices
- Validación de las anteriores expresiones y comparación con *cache oblivious* por medio de simulaciones
- Resultados experimentales en hardware real y comparación con pseudo-LRU

Además, nuestras conclusiones se pueden aplicar con los siguiente beneficios inmediatos:

- En general, solo son necesarias dos vías en la cache de datos y unos pocos conjuntos para la transposición de matrices, de modo que el resto de vías en una cache de datos asociativa pueden ser apagadas, con su correspondiente ahorro de energía y sin ninguna consecuencia negativa
- El último nivel de caches puede proporcionar solo dos vías para la transposición de matrices, evitando la contaminación innecesaria a otros procesos sin un impacto negativo
- Ofrece resultados de aciertos-fallos predecibles (y óptimos) para la versión *tiling* de la transposición de matrices (obligatoria en las aplicaciones de tiempo real) en una cache LRU

El resto del articulo esta organizado de la siguiente manera: La Sección II recoge el trabajo relacionado con la transposición de matrices. La Sección III estudia las tasas de acierto ideales en datos, independientemente de la cache de datos y el algoritmo de transposición. A continuación en la Sección IV analizamos los requisitos para alcanzar las tasas de aciertos ideales (previamente estudiados) para la versión *tiling* de la transposición de matrices. Nuestros resultados se presentan en la Sección V, incluyendo la validación de las conclusiones anteriores, su comparación con el algoritmo *cache oblivious* y su experimentación en hardware real. Finalmente expondremos nuestras conclusiones en la Sección VI.

II. Trabajo relacionado

En esta sección vamos a presentar los trabajos relacionados con el análisis del algoritmo de transposición de matrices, y las transformaciones *tiling* y *cache oblivious*.

En *Cache-Efficient Matrix Transposition* [1] se describen varios algoritmos para la transposición de matrices y comparan su rendimiento usando tanto simulación como los tiempos de ejecución en un sistema basado en Sun UltraSPARC II. Sus simulaciones muestran que mientras que el algoritmo *cache oblivious* alcanza el menor número de fallos de cache para dimensiones pequeñas de matrices, para grandes dimensiones sus resultados son los peores. Además su tiempo de ejecución muestra que en la mayoría de los casos el algoritmo *cache oblivious* es, de manera significativa, más lento que otros algoritmos de transposición. El trabajo sugiere que el bajo rendimiento de este algoritmo está relacionado con la asociatividad de la cache, aunque no exploran en más profundidad esta relación.

En *Cache Oblivious Matrix Transposition: Simulation and Experiment* [4] se explora en más profundidad el algoritmo *cache oblivious* de transposición de matrices con la intención de racionalizar los resultados de Chatterjee y Sen [1]. En este articulo estudian su rendimiento, respecto a los fallos de cache, a través tanto de la simulación como the el uso de contadores hardware en dos sistemas Sun UltraSPARC diferentes. Como en nuestro trabajo, ellos también comparan los algoritmos *tiling* y *oblivious*, pero centrándose en el comportamiento de *oblivious*. Sin embargo los comparan con una configuración de cache y dimensión de *tile* específica, variando solamente la dimensión de la matriz a transponer. Sus resultados muestran que las características de fallos en cache del algoritmo tienen una estructura significativa que depende de la configuración de cache y de la dimensión de la matriz. En sus resultados no son capaces de concluir cuando el rendimiento del algoritmo *cache oblivous* es mejor o peor, solamente que, en general, incrementar la asociatividad de la cache es beneficioso.

En *The Cache Performance and Optimizations of Blocked Algorithms* [2] se centra en optimizar el rendimiento de la cache a través de las transformaciones de *blocking* (*tiling*). Se centran primero en descubrir el comportamiento de las caches bajo las transformaciones de *blocking* y entonces mejorar su rendimiento a través de técnicas software y/o hardware. Analizan la versión *blocking* del algoritmo de multiplicación de matrices y concluyen que el rendimiento de la cache es muy dependiente del tamaño del problema y de bloque (*tile*). Afirman que hay una gran sensibilidad de las tasas de fallos con respecto a la dimensión de la matriz. Sin embargo, nosotros podemos afirmar que esto no se pude asumir para todos los algoritmos de *blocking*, como mostramos en nuestro trabajo.

En *An Experimental Comparison of Cache-oblivious and Cache-conscious Programs* [5] se comparan de manera experimental programas con algoritmos *cache oblivious* y *cache conscious* (a los que han aplicado transformaciones *tiling*) para programas de multiplicación y transposición de matrices. Plantean que coste tiene el uso de programas *cache oblivious* por su habilidad para adaptarse automáticamente a la jerarquía de memoria. Concluyen que incluso los programas *cache oblivious* con altas optimizaciones tienen un rendimiento significativamente peor que sus correspondientes programas *cache conscious*. Sin embargo, no analizan ningún rango de configuraciones de cache, debido a las limitaciones de la experimentación, o un rango de dimensiones de *tiles*, como hacemos en este trabajo.

Algorithm 1 TransposeMatrix($Matrix$, N): Transpone una matriz $N \times N$

```
1: for index1 = 0 to index1 < N do          # N iteraciones
2:     for index2 = index1 + 1 to index2 < N do   # N iteraciones
3:         Swap(index1, index2, Matrix, N)
4:     end for
5: end for
```

Algorithm 2 Swap($index1$, $index2$, $Matrix$, N): En una área de memoria $N \times N$, intercambia $Matrix[index1, index2]$ y $Matrix[index2, index1]$

```
1: elem1 ← index1 × N + index2
2: elem2 ← index2 × N + index1
3: temp ← Matrix[elem1]                     # Lectura de memoria
4: Matrix[elem1] ← Matrix[elem2]            # Lectura y escritura en memoria
5: Matrix[elem2] ← temp                     # Escritura en memoria
```

III. Tasas ideales de aciertos en cache de datos para la transposición de matrices

La tasa de aciertos en cache de datos es el porcentaje de accesos a datos que dan lugar a aciertos en cache. Para estudiar el caso ideal, asumimos que la cache tiene líneas ilimitadas, vacías inicialmente, y un tamaño de línea lo suficientemente grande para contener L elementos de la matriz a transponer ($L \in \mathbb{N}$). Con esta cache de datos solo los accesos que carguen contenido por primera vez darán lugar a fallo. Esto es, solo tendremos en cuenta los fallos obligatorios para nuestras estimaciones de tasas de aciertos. Por lo tanto, el siguiente análisis es independiente del algoritmo de transposición de la matriz. No obstante, los algoritmos 1 y 2 sirven como referencia.

Consideramos una matriz $N \times N$ a transponer, con sus elementos almacenados por filas. También asumimos que la matriz resultante sobrescribirá a la matriz de entrada y no se usarán otras estructuras de memoria, aparte de los registros.

Teniendo en cuenta las consideraciones anteriores, podemos delimitar fácilmente la tasa de aciertos que podemos esperar. Todos los elementos, a excepción de los de la diagonal ($N^2 - N$), se acceden dos veces, uno para leerlos y otro para escribirlos, así que hay un total de $2(N^2 - N)$ accesos. La matriz se compone de $N^2 - N$ elementos accedidos, así que al menos habrá $(N^2 - N)/L$ fallos, que son los obligatorios. Al dividir por L estamos diciendo de manera implícita que los elementos de la diagonal no comparten línea de cache con otros elementos. Aunque asumir esto no es realista (podría complicar los accesos indexados) nos proporciona la siguiente asíntota para la tasa de aciertos en datos:

$$1 - \frac{(N^2 - N)/L}{2(N^2 - N)} = 1 - \frac{1}{2L} \tag{1}$$

Para calcular la tasa de aciertos ideal en datos se debe considerar la relación entre el tamaño de la matriz y de la línea de cache.

A. *Matriz con tamaño de fila múltiplo del tamaño de la línea de cache ($r = 0$)*

Asumimos que la matriz a trasponer esta almacenada en memoria por filas, incluyendo los elementos de la diagonal. También asumimos una matriz $N \times N$ y una línea de cache de datos que contiene L elementos consecutivos de la misma fila, siendo N múltiplo de L. Usando la notación de *módulo*, $r = (N \bmod L) = 0$.

El número total de accesos es el mismo que en la ec. 1. El número de fallos obligatorios en este caso es $(N/L)N$, esto es el número de líneas de cache necesarias para almacenar una fila por el número de columnas. Por lo tanto, la tasa de aciertos ideal es:

$$1 - \frac{N^2/L}{2(N^2 - N)} = 1 - \frac{1}{2L} - \frac{1}{2L(N-1)} \tag{2}$$

B. *Matriz con relleno, con solo un elemento en la última línea de cache ($r = 1$)*

Cuando el número de elementos en una matriz almacenada por filas (N) no es múltiplo del número de elementos de la línea de cache (L) decrece el rendimiento. Una buena y conocida transformación de los datos es el relleno (*padding*),con la cual cada fila en la matriz se extiende artificialmente para completar la línea de cache. Esto permite que el primer elemento de la fila coincida siempre con el primer elemento de la línea de cache. En esta sección asumimos $r = (N \bmod L) = 1$, así que el último elemento de cualquiera de las filas de la matriz no comparte línea de cache con otros elementos. Así, cada fila de la matriz se completa con $L - 1$ elementos de relleno. Por lo cual cada fila supone $(N \operatorname{div} L) + r$ fallos obligatorios asumiendo división entera, y $N/L - r/L + r$ fallos obligatorios si asumimos división real. También hay que tener en cuenta que, para $r = 1$, la última fila es un caso especial ya que su último elemento pertenece a la diagonal y por lo tanto no se accede a él. La tasa ideal de aciertos con $r = 1$ es:

$$1 - \frac{(N/L - 1/L + 1)N - 1}{2(N^2 - N)} = 1 - \frac{1}{2L} - \frac{1}{2N} \tag{3}$$

C. *Matriz con relleno, con varios elementos en la última línea de cache ($r > 1$)*

Asumiendo el mismo relleno que antes, para $r = (N \bmod L) > 1$ el número de fallos obligatorios para cada fila es $N/L - r/L + 1$. En este caso, la tasa ideal de aciertos con $1 < r < L$ ($r \in \mathbb{N}$) es:

$$1 - \frac{(N/L - r/L + 1)N}{2(N^2 - N)} = 1 - \frac{1}{2L} - \frac{1 + L - r}{2L(N-1)} \tag{4}$$

En la Figura 1 se muestran las tasas de aciertos ideales para $L = 4$ elementos cuando incrementamos el tamaño de la matriz. Cada línea señala el correspondiente r para cada N, es decir, $r = 0$ corresponde con los N múltiplos de L (ec. 2), $r = 1$ describe la ec. 3, y $r = 2$ y $r = 3$ representan la ec. 4. Todas estas líneas tienden a la asíntota $1 - \frac{1}{2L} = 0{,}875$ (ec. 1).

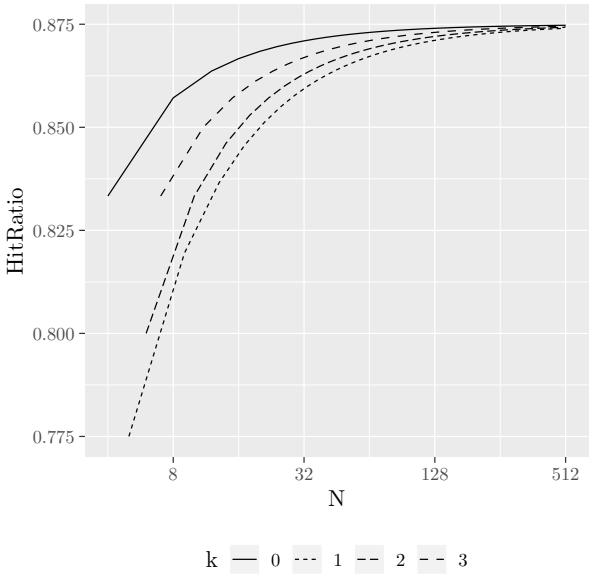


Fig. 1: Tasas de aciertos ideales para $L = 4$ elementos con diferentes valores de r ($r < L, r \in \mathbb{N}$).

Algorithm 3 Tiling(Matrix,N,T)

```
1:  for i = 0 to i < N; i+ = T do
2:      for j = 0 to j < i; j+ = T do
3:          for k = i to k < i + T; + + k do
4:              for l = j to l < j + T; + + l do
5:                  Swap(k,l,N,Matrix)
6:              end for
7:          end for
8:      end for
9:      for k = i to k < i + T − 1; + + k do
10:         for l = k + 1 to l < i + T; + + l do
11:             Swap(k,l,N,Matrix)
12:         end for
13:     end for
14: end for
```

IV. Análisis de óptimos para el algoritmo *tiling* de la transposición de matrices en LRU

En las secciones previas hemos determinado las tasas de aciertos ideales para la transposición de matrices. En esta sección analizaremos la tasa de aciertos óptima para la transposición de matrices asumiendo una cache asociativa por conjuntos LRU. También asumimos que la matriz se traspone por bloques (*tiles*). Esto se corresponde con el algoritmo *tiling* de la transposición (Algoritmo 3), el cual divide la matriz a transponer en pequeños submatrices que encajen en la cache [2]. Por tanto, los *tiles* se transponen completamente en orden secuencial, por lo que cada *tile* permanece en cache mientras se procesa, evitando los fallos por capacidad. Esencialmente esta transformación implica añadir bucles externos al código original manteniendo la secuencia de accesos localizada en los *tiles* procesados. En este análisis nos centramos en la relación entre el tamaño de la matriz ($N \times N$ elementos), el *tile* ($T \times T$ elementos) y la configuración de la cache LRU (S conjuntos, W vías, y líneas con capacidad para L elementos).

También asumimos un relleno extra adicional al de cada fila de la matriz que completan la línea de cache (Sección III-B). Si es necesario se añade a cada fila un relleno adicional de *desplazamiento de fila* equivalente a una línea de cache de tamaño L para asegurar que los primeros elementos en filas consecutivas son mapeados en conjuntos distintos. Esta es una transformación de relleno sobradamente conocida también que pretende minimizar los fallos por conflicto. Dividimos nuestro análisis en tres casos dependiendo de la relación entre el tamaño de *tile* y de línea de cache, estos son, $T = L$, $T > L$, y $T < L$.

A. Transposición óptima de la matriz para $T = L$

Asumiendo la versión *tiling* de la transposición de matrices (Algoritmo 3), la matriz es procesada *tile* por *tile*. En cada *tile*, T elementos horizontales son intercambiados con T elementos verticales. Con $T = L$, el relleno (Sección III-B) garantiza que todos los elementos horizontales caben en una única línea de cache. Además el relleno de desplazamiento de fila (Sección IV) asegura que cada elemento vertical se mapea en un conjunto de cache distinto y por lo tanto no habrá conflictos entre ellos siempre y cuando $S \geq T$. Por lo tanto el único conflicto posible que puede ocurrir implicaría a la línea de cache que contiene a los elementos horizontales, y a la que contiene a los verticales. Este conflicto potencial se evita con cualquier cache LRU con, al menos, 2 vías.

Por lo tanto, la tasa de aciertos ideal para la transposición de matrices (Sección III) se alcanza con una cache LRU cuando $T = L$, $S \geq L$, $W \geq 2$, independientemente del tamaño de matriz. En otras palabras, la mínima asociatividad de cache necesaria con $T = L$ y $S \geq L$ para alcanzar la tasa de aciertos ideal es 2:

$$\text{mín}(W \mid S \geq L) = 1 + 1 \tag{5}$$

En general las caches tienen un gran número de conjuntos así que los modelos de la ec. 5 son el caso general. Para considerar todas las posibilidades analíticas vamos a analizar el caso $S < L$. Con $S < L$ pueden aparecer nuevos fallos por conflicto entre líneas que contienen elementos verticales y horizontales para transponer. Por lo tanto, para alcanzar la tasa de aciertos serán necesarias más vías que absorban estos fallos por conflicto. Específicamente, son necesarias T/L vías para evitar los conflictos entre los elementos verticales más una vía adicional para evitar los conflictos con la línea de cache que contiene los elementos horizontales a transponer. Con estas condiciones la mínima asociatividad necesaria para alcanzar la tasa de aciertos ideal es:

$$\text{mín}(W \mid 1 < S < L) = \left\lceil \frac{T}{S} \right\rceil + 1 \tag{6}$$

Por último, las caches de mapeo directo ($S = 1$) constituyen un caso particular ya que la asociatividad debe ser lo suficientemente grande para proporcionar todas las líneas de cache necesarias para procesar cada fila/columna en un *tile*, es decir, $T + 1$ líneas de cache. Sin embargo, el orden específico de

la política de reemplazo LRU cuando *todas* las filas/columnas en un *tile* están siendo procesadas pueden necesitar una línea de cache adicional. Por ejemplo, se necesitan 4 líneas para completar la transposición de un *tile* con $T = L = 2$ y $S = 1$ sin fallos por conflictos. Así, la mínima asociatividad de cache necesaria con $T = L$ y $S = 1$ para alcanzar siempre la tasa de aciertos ideal es:

$$\text{mín}(W \mid S = 1) \leq T + 2 \tag{7}$$

B. Transposición óptima de la matriz para $T > L$

En general la caches de datos tienen una configuración fija dependiendo del procesador asociado y solo el último nivel de caches permiten alguna configuración, tal como poder restringir el número de vías utilizables por cierto proceso (Intel Skylake SP Gold 5120). Por otro lado el software se puede ajustar fácilmente para poder aprovechar la configuración específica del sistema. Por lo tanto, se debe fijar un *tile* de tamaño $T = L$ para obtener la tasa de aciertos ideal ya que esta es la configuración que permite alcanzar la tasa de aciertos ideal con los requisitos mínimos de cache de datos. De todos modos, la tasa ideal de aciertos también se puede lograr con $T > L$ con caches de datos más grandes. En este caso se necesitan $\frac{T/L}{S}$ líneas de cache adicionales para contener los elementos horizontales a transponer. Dependiendo de la relación entre los parámetros LRU y el tamaño de la matriz la tasa de aciertos ideal se alcanzará con un número ligeramente distinto de vías. Por ejemplo, las matrices con dimensión N que no sea múltiplo de 2 necesitan una vía menos. En cualquier caso la mínima asociatividad necesaria para alcanzar la tasa de aciertos ideal es:

$$\text{mín}(W) \leq \left\lceil \frac{T}{S} \right\rceil + \left\lceil \frac{T/L}{S} \right\rceil + 1 \tag{8}$$

C. Transposición óptima de la matriz para $T < L$

Igual que en la anterior, en esta sección analizamos la relación entre el tamaño de la matriz y los parámetros de una cache LRU para alcanzar la tasa de aciertos ideal en la transposición de matrices, en este caso asumiendo $T < L$. Primero señalar que esta es una suposición muy poco práctica ya que implica que al procesar un *tile* $T \times T$ se carga en cache más contenido que el estrictamente necesario para procesar el *tile*. Por lo tanto para alcanzar la tasa de aciertos ideal este contenido que no ha sido usado no debería ser expulsado hasta que se use. En otras palabras, se necesita una cache lo suficientemente grande para evitar los fallos por capacidad. En concreto el número de líneas de cache ($S \times W$) deber ser mayor que el doble del número de columnas de la matriz ($2N$). Con un tamaño de *tile* tan inadecuado, el número de vías necesarias para alcanzar la tasa de aciertos ideal depende del tamaño de la matriz de la siguiente manera:

$$W = \frac{2N}{S} + 1 \tag{9}$$

V. Resultados

En esta sección vamos a validar las expresiones analíticas previas por medio de extensivas simulaciones. Calculamos las tasas de acierto en la cache de datos para diferentes configuraciones de cache para un amplio conjunto de tamaños de matrices. Además, se comparan los resultados obtenidos con los del algoritmo *cache oblivious*, demostrando que si el tamaño de *tile* $T \times T$ se establece de manera que T sea equivalente a la línea de cache ($T = L$), *tiling* funciona siempre igual o mejor que *cache oblivious*. Por último, lanzamos varios experimentos en hardware real para estudiar su tiempo de ejecución (y no solo la tasa de aciertos de la cache de datos). Los resultados obtenidos son coherentes con lo que se esperaba.

A. Tasa de aciertos en la cache de datos para la versión tiling de la transposición de matrices

En esta sección vamos a validar la accesibilidad del ratio de aciertos ideal en datos para la versión *tiling* de la transposición de matrices (Section III). Asumimos una cache LRU variando su tamaño de línea L, número de conjuntos S y número de vías W. Las matrices de prueba tienen un tamaño desde 1024×1024 a 2048×2048 elementos, con relleno tal como se describe en la Sección IV.

En la Figura 2 se muestran varias subfiguras que corresponden a diferentes tamaños de línea de cache L. En cada subfigura encontramos un *boxplot* para cada combinación de número de conjuntos (S), vías (W), y tamaño de *tile* (T). Cada *boxplot* recoge la tasa de aciertos de datos para matrices desde 1024×1024 hasta 2048×2048 elementos, esto es, 1025 experimentos. Un *boxplot* es una caja cuyos límites representan el primer y tercer cuartil, con una marca dentro que representa la mediana. Las líneas verticales fuera de la caja muestran la variabilidad fuera de esos cuartiles y los puntos señalan los valores atípicos, es decir, aquellos resultados que no son estadísticamente relevantes. En esta figura la mayoría de los casos muestran solamente una línea horizontal, lo cual significa que todas la matrices coinciden en la misma tasa de aciertos de datos o que sus diferencias son inapreciables. La línea punteada horizontal representa la asíntota de tasa de aciertos $1 - \frac{1}{2L}$ (ec. 1). El color del *boxplot* muestra si todas las matrices alcanzan la tasa de aciertos ideal (Sección III), coloreado en negro o si hay alguna de las matrices (o todas) que no lo alcanzan, coloreado en gris. Se puede ver que siempre que el eje-x (T) coincide con la L de la subfigura el color del *bloxpot* es negro, cumpliendo con las ecuaciones ec. 5, ec. 6, and ec. 7. Cuando se alcanzan los resultados ideales con un tamaño de *tile* subóptimo ($T > L$), el tamaño óptimo de *tile* ($T = L$) también alcanza la tasa ideal de aciertos. Señalar también que las propiedades de la política LRU garantizan que si se alcanza la tasa ideal de aciertos para cierta configuración de cache, se alcanzará también cuando incrementamos el número de conjuntos y/o vías. Así que si la cache es demasiado pequeña o sus parámetros no están proporcionados

Algorithm 4 Transpose(Matrix, N, ind1=0, ind2=N-1)

```
1:  if ind2 − ind1 ≤ 2 then
2:    Matrix_{ind1,ind1+1} ↔ Matrix_{ind1+1,ind1}
3:  else
4:    indhalf ← (ind1 + ind2)/2
5:    Transpose(Matrix, N, ind1, indhalf)
6:    if indhalf < N then
7:      Transpose(Matrix, N, indhalf, ind2)
8:      TransposeSwap(Matrix, N, indhalf,
9:        ind1, ind2, indhalf)
10:   end if
11: end if
```

Algorithm 5 TransposeSwap(Matrix,N,rs,cs,re,ce)

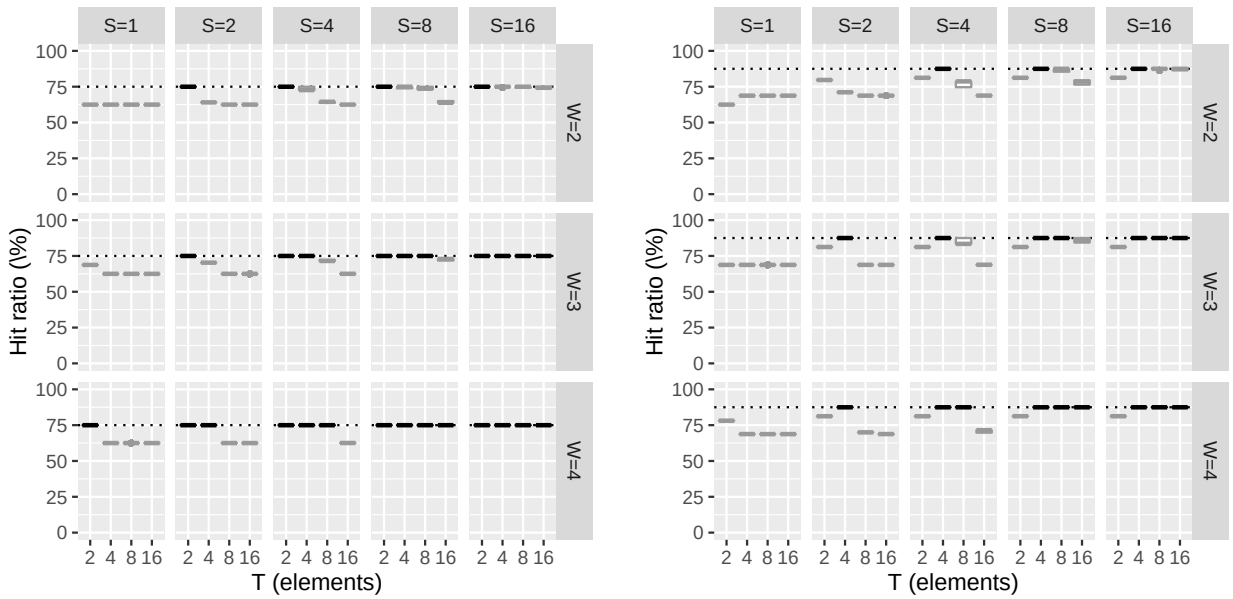
```
1:  if ((re − rs) ≤ 2 and (ce − cs) ≤ 2) then
2:    for r = rs to r < re do
3:      for c = cs to c < ce do
4:        Swap(r,c,N,Matrix)
5:      end for
6:    end for
7:  else
8:    if rs < N then
9:      rm ← (rs + re)/2
10:     cm ← (cs + ce)/2
11:     TransposeSwap(Matrix, N, rs, cs, rm, cm)
12:     TransposeSwap(Matrix, N, rm, cs, re, cm)
13:     TransposeSwap(Matrix, N, rs, cm, rm, ce)
14:     TransposeSwap(Matrix, N, rm, cm, re, ce)
15:   end if
16: end if
```

quizá no se pueda alcanzar la tasa ideal de aciertos. Afortunadamente estos casos no son realistas y las caches de datos existentes son mucho más grandes que aquellas representadas en Figura 2.

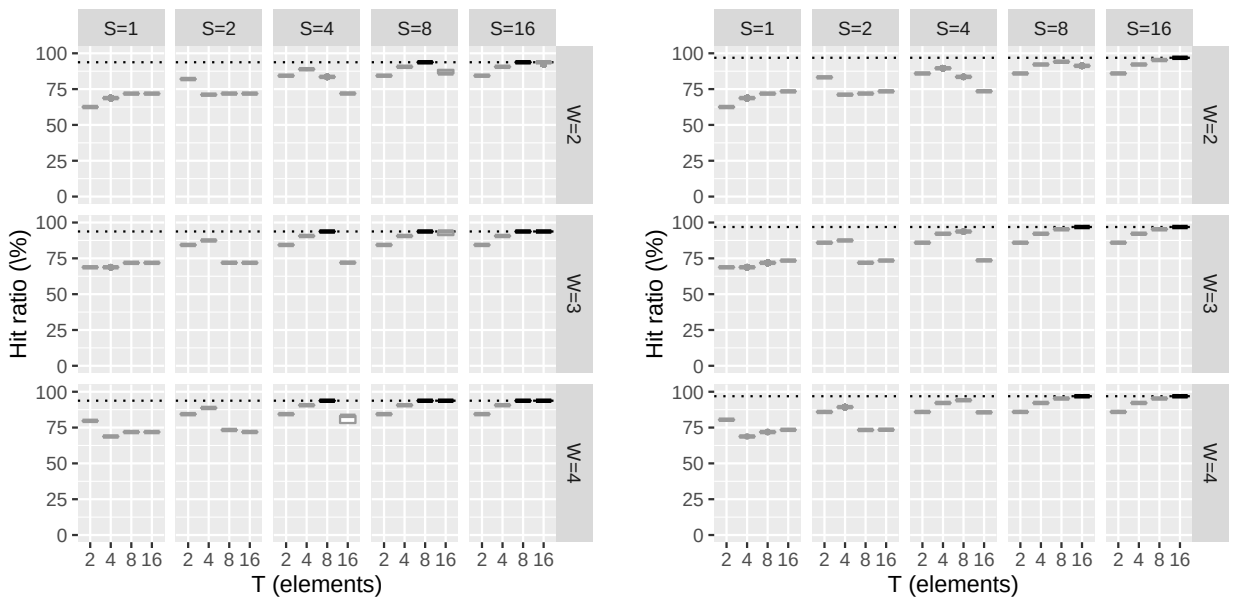
B. *Tasa de aciertos en datos de* tiling *frente a* cache oblivious

Los resultados anteriores muestran que, con el tamaño de *tile* adecuado ($T = L$), la tasa de aciertos ideal se alcanza con tamaños de cache muy pequeños. En vez de usar un algoritmo *tiling*, cache consciente, podríamos usar un algoritmo *cache oblivious*. En esta sección evaluamos los requisitos de la cache de datos para alcanzar la tasa de aciertos ideal con un algoritmo *oblivious* [6] y los comparamos con nuestros resultados de *tiling*. Un algoritmo *cache oblivious* (Algoritmos 4 y 5) puede interpretarse como un algoritmo *tiling* donde la matriz procesada se divide recursivamente hasta alcanzar, en la transposición de matrices, el tamaño 2×2. Por lo tanto probablemente el *tile* encajará en la cache al procesarlo, y el orden impuesto por la recursividad es una variación de *Z-order*, que mantiene la localidad [7]. Por todo esto, la aplicación de este algoritmo no necesita conocer los parámetros de la configuración de cache. Sin embargo, su rendimiento sí que depende de ellos, y predecir cuando funcionara bien o mal no es trivial [4].

Este proceso es problemático cuando la dimensión de la matriz (N) no es potencia de 2, ya que *oblivious* trabaja recursivamente dividiendo la matriz a transponer. Este efecto puede verse en la Figura 3, donde solo las matrices cuya dimensión es potencia de dos alcanzan la tasa de aciertos ideal. Para evitar este efecto, introducimos un *relleno fantasma* en el Algoritmo 4. Este relleno no modifica el mapeo en memoria de la matriz a transponer, solo fuerza al algoritmo a recorrer la matriz como si su dimensión fuera potencia de 2. Por lo tanto, el intercambio de elementos se lleva a cabo solo si son reales, es decir, no se intercambian los elementos "fantasma". Esta transformación tampoco afecta al número de datos accedidos, simplemente cambia la secuencia de accesos. Tras corregir este comportamiento, hemos repetido los experimentos de la Sección V-A para el algoritmo *oblivious*. En la Figura 4 resumimos nuestros resultados para *tiling* (Figura 2), y aquellos para *oblivious*. En ella se muestra las vías mínimas necesarias en cache W para alcanzar la tasa de aciertos ideal para cada combinación de tamaño de línea de cache L y número de conjuntos de cache S, es decir, cuanto menor mejor. Los resultados de *tiling* los podemos encontrar a la izquierda y los *oblivious* a la derecha. Solo se muestran tamaños de *tile* adecuados ($T = L$) ya que aquellos inadecuados ($T > L$, $T < L$) presentan peores resultados, como ya hemos presentado. Las barras de color gris oscuro señalan el mínimo número de conjuntos para alcanzar la tasa de aciertos ideal. Puede verse que *oblivious* necesita el mismo número de vías que *tiling* en caches con el suficiente número de conjuntos ($T = L, S \geq L$). Sin embargo, en caches con menos conjuntos (incluyendo las totalmente asociativas), *oblivious* necesita más vías que *tiling* para conseguir la tasa ideal de aciertos. Esto es debido a que *oblivious* no para la recursividad cuando alcanza el tamaño adecuado de *tile* sino que siempre lo reduce hasta 2×2 elementos. Así que cuanto más grande es el tamaño de línea de cache más contenido innecesario trae de memoria para procesar cada 2×2 *tile*. Este contenido innecesario (el cual se necesita en los siguientes *tiles*) debe permanecer cacheado para lograr la tasa de aciertos de datos ideal. Así que, fijando el número de conjuntos, se necesita un mayor número de vías. Las caches de datos suelen tener un número relativamente grande de conjuntos ($S \geq L$), así que en general tanto la transposición de matrices *cache oblivous* como la *cache consciente* van a alcanzar la tasa de aciertos ideal. Sin embargo, hay que tener en cuenta que las caches son utilizadas por varios procesos al mismo tiempo, así que acceder a ellas tan eficientemente como sea posible sigue siendo importante. Los resultados anteriores nos muestran que con *tiling* necesitamos menos recursos para alcanzar la tasa de aciertos ideal que con *oblivious*. También señalar que los códigos recursivos como el algoritmo *oblivious* suelen ser más lentos que los iterativos debido a las llamadas a funciones y el procesamiento de la pila.



(a) $L = 2$ (b) $L = 4$



(c) $L = 8$ (d) $L = 16$

Fig. 2: Tasa de aciertos de datos para diferentes configuraciones de cache (número de conjuntos S, vías W y tamaño de línea de cache en número de elementos L) y tamaños de *tile* ($T \times T$ elementos), para matrices desde 1024×1024 a 2048×2048 elementos. El color de los *boxplot* indican si todas las matrices alcanzan la tasa ideal de aciertos (negro) o no (gris) (Sección III).

C. Resultados experimentales

En los resultados previos asumimos una cache de datos asociativa por conjuntos. Debido a que construir una política LRU eficiente para una cache de gran tamaño es costoso, la mayoría de las caches usan la política PLRU, que ofrece un comportamiento similar pero es más sencilla de construir [8]. También aunque la tasa de aciertos presentada anteriormente es probablemente el factor más determinante en cuanto al rendimiento otros elementos como el número de instrucciones ejecutadas (dependiente del tamaño de *tile*) también son importantes. Para considerar estos y otros factores (por ejemplo, la prebúsqueda en datos) en esta sección mediremos lo tiempos de ejecución de la transposición de matrices. Para ello hemos ejecutado un programa que calcula 400 transposiciones de una matriz de tamaño 4096×4096, realizando un vaciado de la cache antes de cada una de ellas. El tiempo de ejecución se mide en cada transposición.

Los experimentos han sido llevados a cabo por varias máquinas, a saber, Xeon-L5410 2.33GHz, i7-4810MQ 2.80GHz, Core-2-Quad-Q9550 2.83GHz y i7-2640M 2.80GHz. Todas ellas tienen una cache de instrucciones L1 y una cache de datos L1 de 64 con-

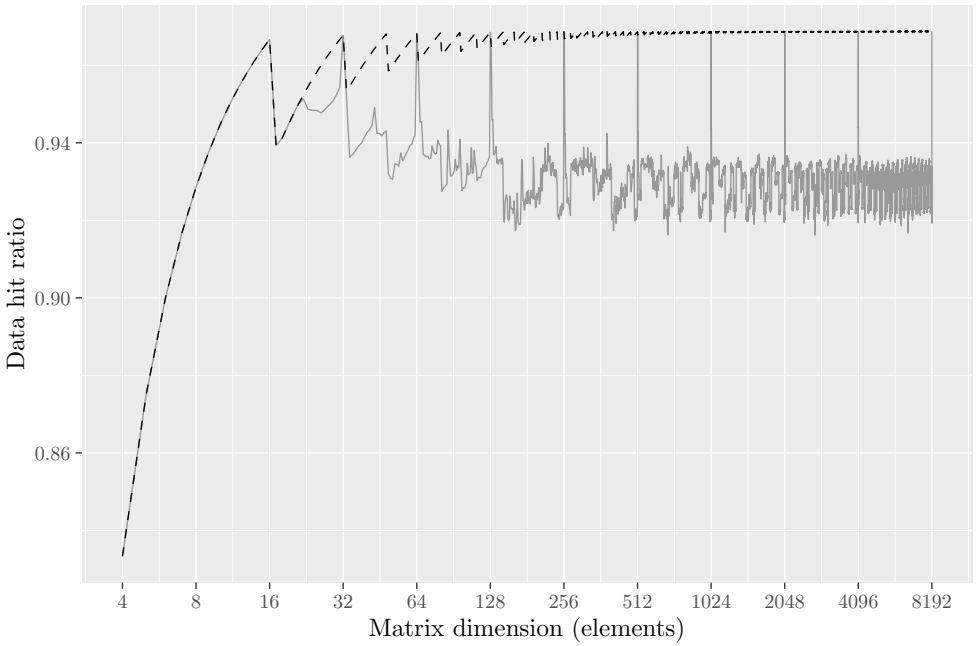


Fig. 3: Tasa de aciertos (*Data hit ratio*) para el algoritmo *oblivious* para tamaños de matriz (*Matriz dimension (elements)*) desde $4x4$ a $8192x8192$ (gris). La tasa de aciertos ideal está representada por la línea negra. Configuración de cache: $L = 16, S = 16, A = 2$

juntos, 8 vías y 64 B por línea, con un tamaño total de 32 KiB. Con este tamaño de línea cada una contiene 8 elementos de la matriz. Para cada máquina ha sido generado un binario diferente usando su compilador disponible (gcc-4.7.0 para i7-4810M y Core-2-Quad-Q9550, Intel C++ Composer XE 2013 para Xeon-L5410 y gcc-4.9.2 para i7-2640), siempre con un nivel de optimización 3. En los experimentos se varía la dimensión del *tile* desde $T = 2$ hasta $T = 1024$.

La Figura 5 muestra el tiempo de ejecución mínimo de transponer una matriz 4096×4096 (de un total de 400 transposiciones por tamaño de *tile*).

El área sombreada muestra las configuraciones de *tile* que alcanzarían la tasa de aciertos ideal en LRU. El *tile* 8×8 corresponde a $T = L$ (tasa de aciertos ideal con los mínimos recursos de cache) *tiles* mayores, hasta 256×256, alcanzan la tasa ideal de aciertos usando más conjuntos/vías (ec. 8). El tiempo se incrementa para LRU fuera del área sombreada y se puede apreciar que PLRU también presenta este comportamiento. En los experimentos con intel-i7, el tiempo de ejecución para *tiles* grandes aumenta pero muy ligeramente. Esto se debe a la prebúsqueda de accesos a datos, que reconoce el patrón de acceso a datos evitando así penalizaciones en tiempo. En el área de la izquierda, los *tiles* son demasiado pequeños para que la prebúsqueda de datos reconozca el patrón de accesos, así que el tiempo necesario para estos *tiles* es bastante mayor. El tamaño de *tile* también afecta al número de instrucciones ejecutadas. Tamaños múltiplos de 8 trabajan con toda la línea de cache así que su rendimiento en general es mejor que otros tamaños. Se puede ver claramente que entre 8 y 16, todos las gráficas muestran un incremento del tiempo de ejecución. Por último, cuanto mayor el tamaño de *tile* menor el número de instrucciones ejecutadas y saltos tomados. Esto es, cuanto menor es el tamaño de *tile* dentro del área sombreada más eficiente es el uso que se hace de la cache, pero aquellos tamaño a la derecha ejecutan menos instrucciones. Por lo tanto, hay una compensación entre los recursos de cache y la instrucciones ejecutadas, y la tendencia que se observa en cuanto al tiempo de ejecución necesario es diferente dependiendo de la máquina dónde se ejecute. En media PLRU presenta un comportamiento similar a LRU con una coste hardware menor (energía y área). En la Figura 5 se puede observar que este comportamiento es similar ya que todo lo previamente planteado para el comportamiento LRU se aplica para PLRU. PLRU trabaja en un árbol binario, dónde cada una una de las dos ramas en un nodo interno se marca como la usada más recientemente que la otra rama. Esto se traduce en una falta importante de predictibilidad en sistemas de tiempo real [9], [3], [10]. Como el problema de la transposición con *tiling* tiene bien definido el orden de acceso, los fallos y aciertos en PLRU pueden simularse fácilmente. Hemos llevado a cabo estas simulaciones para las tres opciones de *tiling*. Para $T = L$ ($T = 8$ en Figura 5),Tanto LRU como PLRU alcanzan la tasa de aciertos ideales en la transposición de matrices. Para $T > L$ ($T > 8$ en Figura 5), LRU proporciona igual o mayor tasa de aciertos que PLRU. Para $T < L$ ($T < 8$ en Figura 5), PLRU proporciona igual o mayor tasa de aciertos que LRU (recordemos que esta es una configuración para *tile* errónea).

VI. Conclusiones

En este artículo estudiamos la tasa de aciertos de datos para el problema de la transposición de ma-

trices. Obtenemos la tasa ideal de aciertos en datos independientemente del algoritmo de transposición y la configuración de cache, calculando sus fallos obligatorios. Después analizamos desde una perspectiva teórica cual es la mejor tasa de aciertos en datos que se puede obtener, dependiendo del tamaño de la matriz, el tamaño de *tile* y la configuración de la cache LRU. Nuestros resultados teóricos muestran que, con una configuración correcta de *tile* (dimensión del *tile* igual al tamaño de la línea de cache), la tasa ideal de aciertos en datos se alcanza con una cache asociativa por conjuntos con solo dos vías y unos pocos conjuntos. Estos resultados se confirman por medio de simulaciones extensivas y también validando los requisitos de cache adicionales cuando el *tile* no se fija correctamente. Este descubrimiento tiene una gran importancia ya que permite ajustar el consumo de energía (apagando los conjuntos y vías innecesarios) o el alto rendimiento (restringiendo a específicos conjuntos/vías la transposición de matrices para que no contamine otros programas) Además, esto permite calcular de manera trivial el peor tiempo de ejecución para entornos de tiempo real.

También comparamos nuestros resultados con aquellos del algoritmo *cache oblivious*, los cuales presentan un buen rendimiento independientemente de la cache. Nuestros resultados muestran que tanto que el algoritmo *tiling* (correctamente configurado) como el algoritmo *oblivious* necesitan dos vías para alcanzar la tasa de aciertos ideal en datos para una cache con un número suficiente de conjuntos.

Si el número de conjuntos en la cache de datos es demasiado pequeño la solución *tiling* supera a la *cache oblivious* que requiere una cache con más vías para alcanzar la tasa ideal de aciertos. Esto también es muy interesante ya que *tiling* proporciona igual o mejor tasa de ratios y al mismo tiempo evita la naturaleza recursiva de los algoritmos *oblivious*.

Finalmente, para probar el rendimiento general y no solo la tasa de aciertos hemos estudiado el tiempo de ejecución de la transposición de matrices en distintas máquinas. Aunque las CPUs actuales usan pseudo-LRU en vez de la politica LRU, nuestros resultados muestran la tendencia que se esperaba. También reflejan otros aspectos como los predictores de acceso a datos, el número de instrucciones ejecutadas y el número de saltos tomados.

Agradecimientos

Este trabajo ha sido financiado por la beca FPU14/02463, el proyecto TIN2016-76635-C2-1-R (AEI/FEDER, UE), el Gobierno de Aragón (Grupo T58_17R) y FEDER 2014-2020 «Construyendo Europa desde Aragón».[1]

[1]Está estrictamente prohibido usar, investigar o desarrollar, de manera directa o indirecta cualquiera de las contribuciones científicas de los autores de este trabajo por cualquier ejercito o grupo armado en el mundo, para propósitos militares o para cualquier uso en contra de los derechos humanos o del medio ambiente, a no ser que se cuente con el consentimiento escrito de todos los autores de este trabajo, o de todas las personas del mundo.

Referencias

[1] S. Chatterjee and S. Sen, "Cache-efficient matrix transposition," in *Proceedings Sixth International Symposium on High-Performance Computer Architecture. HPCA-6 (Cat. No.PR00550)*, Jan 2000, pp. 195–205.

[2] Monica D. Lam, Edward E. Rothberg, and Michael E. Wolf, "The cache performance and optimizations of blocked algorithms," *SIGPLAN Not.*, vol. 26, no. 4, pp. 63–74, Apr. 1991.

[3] J. Reineke, D. Grund, C. Berg, and R. Wilhelm, "Timing predictability of cache replacement policies," *Real-Time Systems*, vol. 37, no. 2, pp. 99–122, Nov. 2007.

[4] Dimitrios Tsifakis, Alistair P. Rendell, and Peter E. Strazdins, "Cache oblivious matrix transposition: Simulation and experiment," in *Computational Science - ICCS 2004*, Marian Bubak, Geert Dick van Albada, Peter M. A. Sloot, and Jack Dongarra, Eds., Berlin, Heidelberg, 2004, pp. 17–25, Springer Berlin Heidelberg.

[5] Kamen Yotov, Tom Roeder, Keshav Pingali, John Gunnels, and Fred Gustavson, "An experimental comparison of cache-oblivious and cache-conscious programs," in *Proceedings of the Nineteenth Annual ACM Symposium on Parallel Algorithms and Architectures*, New York, NY, USA, 2007, SPAA '07, pp. 93–104, ACM.

[6] Matteo Frigo, Charles E. Leiserson, Harald Prokop, and Sridhar Ramachandran, "Cache-oblivious algorithms," in *Proceedings of the 40th Annual Symposium on Foundations of Computer Science*, Washington, DC, USA, 1999, FOCS '99, pp. 285–, IEEE Computer Society.

[7] G. M. Morton, "A computer oriented geodetic data base and a new technique in file sequencing," Tech. Rep., IBM Ltd., Ottawa, Canada, 1966.

[8] Andreas Abel and Jan Reineke, "Measurement-based modeling of the cache replacement policy," in *RTAS*, April 2013.

[9] Christoph Berg, "PLRU cache domino effects," in *Workshop On Worst-Case Execution Time (WCET) Analysis 2006*, Dresden, Germany, July 2006.

[10] Andreas Abel and Jan Reineke, "Reverse engineering of cache replacement policies in intel microprocessors and their evaluation," in *2014 IEEE International Symposium on Performance Analysis of Systems and Software, ISPASS 2014, Monterey, CA, USA, March 23-25, 2014*. 2014, pp. 141–142, IEEE Computer Society.

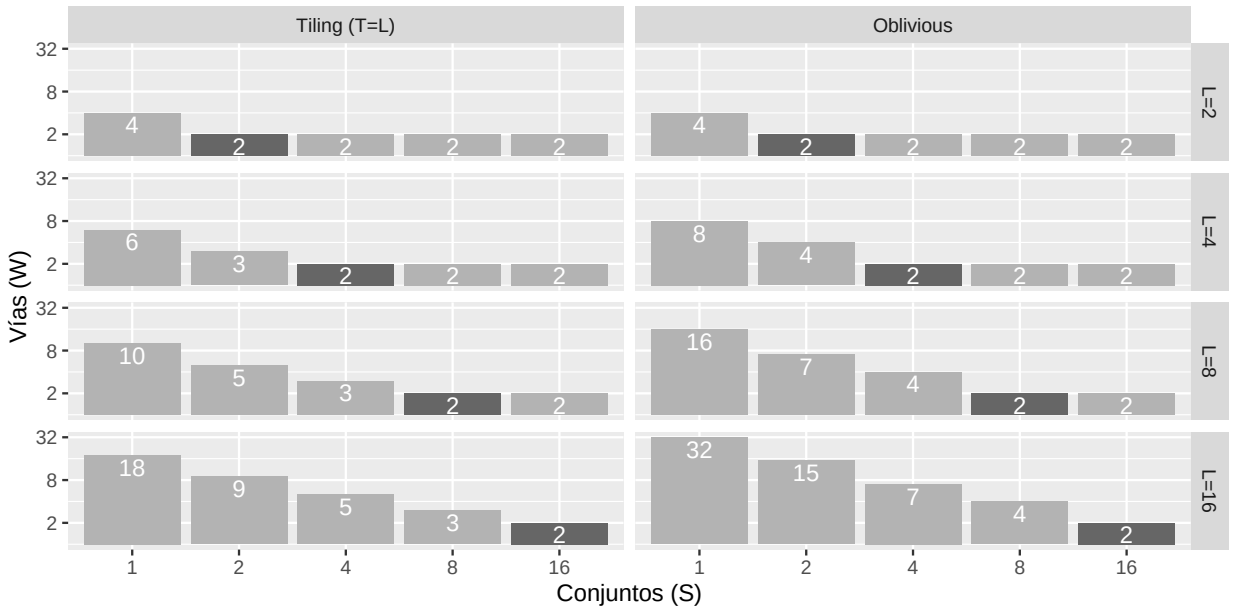


Fig. 4: Dado un *tile* de igual tamaño al tamaño de línea de cache ($T = L$) y un número de conjuntos (S), el mínimo número de vías (W) para alcanzar la tasa de aciertos ideal para cualquier tamaño de matriz para los algoritmos *tiling* y *oblivious*

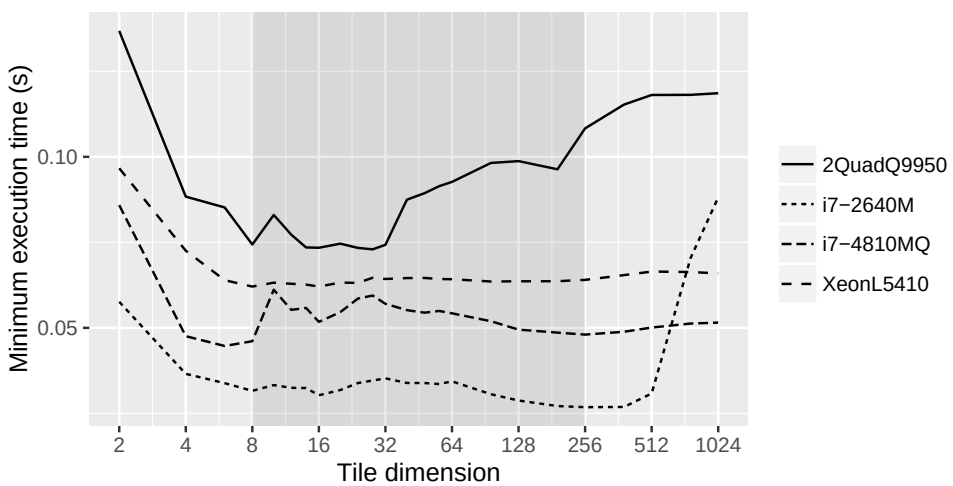


Fig. 5: Mínimo tiempo de ejecución para la transposición de una matriz 4096×4096 de un total de 400 transposiciones.

FOS-Mt: Una Organización Eficiente de Cache para Aplicaciones Paralelas en Procesadores de Bajo Consumo

José Puche, Salvador Petit, María E. Gómez y Julio Sahuquillo[1]

Resumen— **La jerarquía de cache de los procesadores multinúcleo actuales se compone típicamente de tres niveles, desde el más pequeño y rápido nivel de L1 hasta el más lento y de mayor tamaño nivel de L3. Esta jerarquía ha demostrado ser efectiva en procesadores de alto rendimiento gracias a que reduce el tiempo medio de acceso a memoria. Sin embargo, cuando se implementa en dispositivos donde la eficiencia energética es un factor crítico como en sistemas embebidos o de bajo consumo, estas jerarquías convencionales sufren algunas desventajas. Estas desventajas, que están mayormente relacionadas con el desperdicio de área y energía, son las múltiples busquedas en la cache, la presencia de varias réplicas de un mismo bloque en los distintos niveles y la sobredimensión del espacio privado.**

Con el objetivo de abordar estos problemas, en este trabajo proponemos FOS-Mt, una organización de cache destinada a ahorrar energía en multiprocesadores actuales para aplicaciones multihilo. La jerarquía de cache de FOS-Mt está formada únicamente por dos niveles: un primer nivel L1 situado junto al núcleo de cómputo, y un segundo nivel, accesible por todos los núcleos, que forma un espacio agregado de cache unificando los niveles subsiguientes. Este segundo nivel se encuentra dividido en múltiples y pequeños búferes, que se asignan dinámicamente a los hilos en ejecución. Los búferes que no se asignan a ningún hilo se mantienen apagados para ahorrar energía.

Los resultados obtenidos muestran que FOS-Mt reduce significativamente tanto la energía estática como dinámica consumida con respecto a otras organizaciones de cache. Además, comparada con una técnica de ahorro de energía ampliamente conocida como Cache Decay, nuestra propuesta consigue una mejora del Energy Delay Product de un 19.3% de media.

Palabras clave— **Arquitectura, procesador, cache, óptica, fotónica, NUCA, NUMA, redes on-chip**

I. Introducción

LOS procesadores multinúcleo se han convertido en un componente ubicuo en prácticamente todos los dispositivos actuales. Sin importar el uso del dispositivo para el que el procesador es implementado, el rendimiento de cualquier procesador multinúcleo habitual depende en gran medida del rendimiento del subsistema de memoria y, particularmente, de la jerarquía de cache. Esta jerarquía, típicamente compuesta por tres niveles, tiene el objetivo de ofrecer un buen balance entre rapidez de acceso a los datos y reducción del número de accesos a memoria principal.

La jerarquía de cache típicamente ocupa una proporción de área importante en la superficie del procesador, llegando incluso en algunos casos a suponer hasta un 50% de la misma. Esto se traduce directamente en un consumo significativo de energía, lo que supone una preocupación en el diseño de los procesadores actuales, especialmente en dispositivos de bajo consumo y embebidos. Existen trabajos que proponen apagar individualmente líneas de cache [1] o reducir su voltaje [2]; sin embargo, la implementación práctica de estos trabajos es difícil ya que introducen numerosos problemas en la circuitería de la cache.

En este trabajo se propone FOS-Mt (*Flat On-chip Storage for Multithreaded applications*), una nueva organización de cache que tiene como objetivo reducir el consumo de energía en procesadores de bajo consumo que ejecutan aplicaciones multihilo. La jerarquía de cache de FOS-Mt está formada únicamente por dos niveles: un primer nivel L1 situado junto al núcleo de cómputo, y un segundo nivel, accesible por todos los núcleos, que forma un espacio agregado de cache unificando los niveles subsiguientes. Este segundo nivel se encuentra dividido en múltiples y pequeños búferes, que se encuentran apagados inicialmente y se asignan dinámicamente a los hilos durante la ejecución. En nuestra propuesta, los búferes son la unidad mínima de ahorro de energía, lo que facilita la implementación hardware con respecto a trabajos anteriores.

El resto de este trabajo se distribuye de la siguiente forma. En la sección II se hace una recopilación de trabajo relacionado con la cuestión, prestando especial atención a otras propuestas focalizadas en el ahorro de energía en la jerarquía de cache. La sección III presenta las principales características de FOS-Mt, la organización de cache propuesta en este trabajo. Las secciones IV y V describen el marco experimental utilizado y muestran los resultados obtenidos. Finalmente, la sección VI recoge las conclusiones del presente trabajo.

II. Trabajo relacionado

En esta sección se muestran propuestas anteriores relacionadas con el presente trabajo. Se han dividido los trabajos estudiados en dos categorías, ambas relacionadas con los principales temas tratados en este paper: ahorro de energía en corrientes de fuga producidas en caches, y propuestas de jerarquías de cache eficientes energéticamente.

Ahorro de energía en la memoria cache. El consumo de energía ha sido un problema importante

[1]Departamento de Ingeniería de Sistemas y Computadores, Universitat Politècnica de València, e-mail: jopucla@gap.upv.es

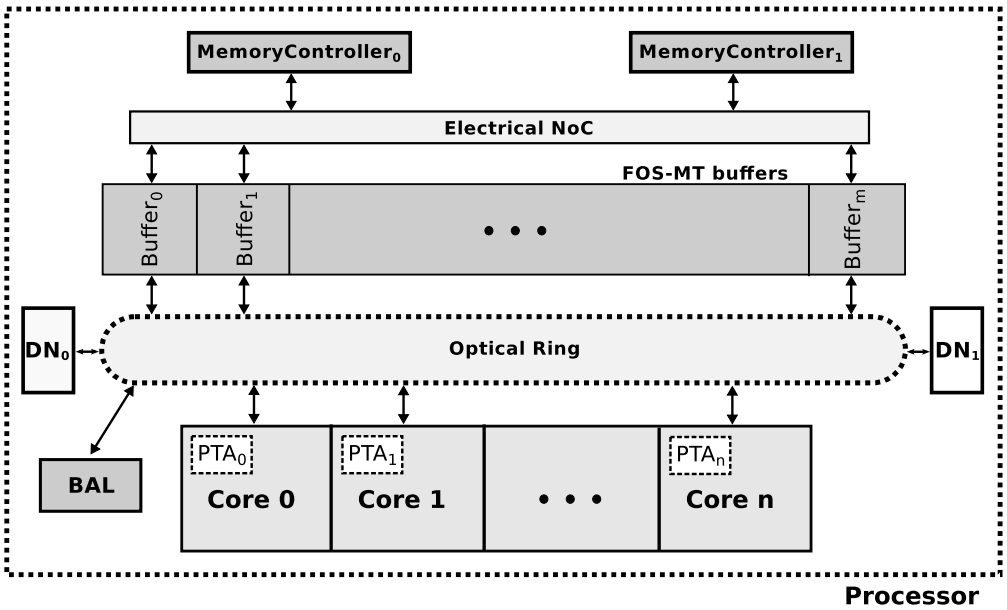


Fig. 1. Diagrama de bloques de FOS-MT.

en el diseño de las caches dado que éstas ocupan un porcentaje significativo de la superficie del procesador. En este sentido, un amplio estudio de ciertas técnicas y propuestas para abordar este problema se recoge en [3].

Uno de los enfoques que más se ha seguido para tratar con el ahorro de energía en caches ha sido reducir la energía consumida por corrientes de fuga en la cache [4]. En esta línea, en [2] los autores proponen las *drowsy caches*, donde diferentes líneas de la cache se establecen en modo de bajo consumo de acuerdo a diferentes criterios. En [5], Powell et al. proponen *Gated-Vdd*, una propuesta que no solo modifica el voltaje suministrado a las celdas de SRAM sino que además reconfigura dinámicamente el tamaño de la cache en función de las necesidades de la aplicación. Finalmente, Kaxitas et al. proponen *Cache Decay* [1], un propuesta que invalida y apaga las líneas de cache cuando se estima que es improbable reutilizar el dato almacenado.

Organizaciones de cache eficientes y adaptativas. Con el objetivo de mejorar la eficiencia energética y el rendimiento de la jerarquía de cache, se ha propuesto una gran variedad de técnicas destinadas tanto caches compartidas [6], [7] como a caches privadas [8], [9], [10]. Perteneciente al primer grupo, en [6], Qureshi y Patt distribuyen y asignan diferentes vías a las aplicaciones e hilos en ejecución. Ubicados en el segundo, los enfoques de caching cooperativo como [8], [9], [10] toman prestado espacio de cache los núcleos colindantes, unificando las caches de L2 privadas y dividiéndolas en espacios privados y compartidos.

Existen además numerosas propuestas aplicadas a caches compartidas y distribuídas que se han focalizado en incrementar el rendimiento y la eficiencia energética mediante diferentes enfoques de organizaciones NUCA [11], [12], [13], [14], [15]. Sin embargo, estos trabajos sufren inconvenientes propios de jerarquías rígidas bien conocidos como son el excesivo movimiento de datos y el *cache trashing*.

III. Flat On-Chip Storage for Multithreaded Applications

FOS-Mt es una arquitectura que rediseña la jerarquía de memoria para mejorar su eficiencia energética y habilitar la implementación práctica de mecanismos de ahorro de energía de granularidad amplia. Mediante la unificación de los niveles de cache L2 y siguientes en un banco común de búferes se consigue comprimir el área dedicada a la jerarquía, que ha sido tradicionalmente uno de los mayores retos en el diseño de procesadores de bajo consumo. Estos procesadores no cuentan ni con el presupuesto energético ni con el tamaño suficientes como para implementar grandes caches de L3 con las que reducir los fallos de capacidad, por lo que administrar correctamente el espacio disponible es un tarea clave en estos sistemas.

La Figura 1 muestra el diagrama de bloques de la arquitectura propuesta. Sus principales componentes son el banco de búferes, las estructuras de directorio que garantizan la coherencia (etiquetadas como DN y PTA), el nodo especial BAL (*Buffer Allocation Logic*) y la red de interconexión.

Los búferes de FOS-Mt se organizan en un banco de elementos que son asignados individualmente a los núcleos de ejecución. Esta organización implica que la latencia de acceso varía dependiendo del búfer que está siendo accedido por un núcleo determinado, lo que puede afectar al rendimiento del sistema. Para abordar este problema, en este trabajo utilizamos una red óptica *on-chip* que ofrece una latencia de acceso prácticamente homogénea a todos los búferes. Obsérvese, sin embargo, que cualquier red de interconexión que satisfaga este requisito es adecuada para ser implementada en FOS-Mt.

FOS-Mt distribuye dinámicamente el espacio de cache disponible de acuerdo con las necesidades de las aplicaciones o hilos en ejecución. Esto sig-

nifica que esquemas de asignación y direccionamiento estáticos como por ejemplo el acceso entrelazado no son válidos en esta arquitectura. Para resolver el problema del acceso a los búferes así como para permitir una gestión flexible de los mismos, FOS-Mt desacopla las etiquetas de los búferes de sus correspondientes datos, y las ubica en dos nuevas estructuras: los *Directy Nodes* (DNs) y las *Private Tag Arrays* (PTAs).

Estas estructuras se encargan de almacenar las etiquetas desacopladas y la información de coherencia, así como de gestionar el acceso a los búferes de datos alejados del núcleo. Las PTAs se encargan de replicar, en cada núcleo, un subconjunto de las etiquetas correspondientes a los bloques accedidos. Los PTAs, a su vez, están compuestos por dos tipos de subestructuras: varios *Buffer Tag Arrays* (BTAs) y una única *Shared Tags Table* (STT). Los BTAs almacenan las etiquetas de los bloques almacenados en los búferes de datos asignados al núcleo, mientras que la STT actúa como una pequeña cache que se utiliza para detectar rápidamente en qué búfer se encuentra un bloque compartido y que ha sido previamente accedido por otro núcleo.

Por otro lado, los DNs conforman una estructura distribuida ubicada fuera de los núcleos que almacena el total de las etiquetas y la información de coherencia correspondiente a los datos ubicados en los búferes activos. Los DNs son así la estructura a la que se accede cuando el bloque no ha sido encontrado ni en los BTAs de los búferes locales del núcleo ni en la pequeña STT, que únicamente retiene un número reducido de entradas (i.e. 1024) para bloques compartidos. Cuando un bloque no es localizado en ninguna de estas 3 estructuras, se realiza un acceso a memoria y se trae el bloque al búfer activo correspondiente.

Adicionalmente, se necesita de una *Buffer Allocation Logic* (BAL) para mantener un registro acerca de los búferes que se encuentran asignados en cada momento a los núcleos en ejecución. Esta lógica se conecta al anillo óptico mediante un nodo especial, etiquetado en la Figura 1 como BAL.

Para finalizar, el último componente expuesto en la Figura 1 es la red óptica de interconexión. En este trabajo se ha seleccionado un anillo óptico extraído del estado del arte que cumple dos requisitos para permitir el adecuado funcionamiento de FOS-Mt: i) ofrece latencias lo suficientemente reducidas como para no introducir retardos en el acceso a los búferes; y ii) la variabilidad entre las latencias de acceso a diferentes búferes y desde diferentes núcleos se mantienen un margen lo suficientemente estrecho como para no afectar negativamente a las prestaciones del sistema. Por estas razones, se ha utilizado este tipo de red para comunicar el resto de componentes descritos anteriormente.

A. Gestión de búferes en FOS-Mt

La gestión de búferes en FOS-Mt cubre tanto la asignación como la liberación de los mismos. La primera tarea se realiza mediante el Algoritmo de Asignación de Búfferes (AAB), que asigna búferes a los núcleos de cómputo cuando se espera que ésta asignación tenga un efecto lo suficientemente positivo en el rendimiento de los mismos. La segunda tarea se lleva a cabo mediante un simple Mecanismo de Liberación que desasigna un búfer de un núcleo basándose en criterios de tiempo y actividad.

Algorithm 1 Algoritmo de Asignación de Búferes.

Entradas:

n: número de intervalo;
b: cantidad de búferes asignada actualmente al núcleo;
K: categoría de MPKI del hilo;
$\{Thr_x\}$: umbrales del algoritmo.

1. **Medición de Rendimiento de Búferes.** Por cada hilo en ejecución:

$$BTAs\_MPKI_{n-1}(b),\ BTAs\_MPKI_{n-1}(b+1),\ BTAs_{hits} \text{ y } STT/DN_{hits}.$$

2. **Filtrado.** Por cada hilo en ejecución:

si ($MPKI_{n-1} < Thr_{min}$) **o** ($BTAs_{hits} < Thr_{PH}$)

Saltar Paso 3 and finalizar.

3. **Análisis de Rendimiento.** Por cada hilo en ejecución:

$$MPKI_{dec\_rate} = \frac{BTAs\_MPKI_{n-1}(b+1)}{BTAs\_MPKI_{n-1}(b)} - 1,$$

$$BTAs_{weight} = \frac{BTAs_{hits}}{BTAs_{hits}+STT/DN_{hits}}.$$

si ($BTAs - MPKI_{dec\_rate} > K \times Thr_{MPKI}$) **y** ($BTAs_{weight} > K \times Thr_{weight}$)

Solicitar un nuevo búfer.

El principal reto del algoritmo AAB consiste en estimar la mejora de rendimiento que espera obtener un determinado hilo si se le asigna un nuevo búfer. Para este propósito, FOS-Mt almacena unas etiquetas *sombra* adicionales en los PTAs, que se utilizan para estimar el número de misses obtenidos con un búfer adicional. Para reducir la sobrecarga de estas etiquetas, solo se replican 32 conjuntos por cada búfer del PTA, tal y como se hace en otros trabajos anteriores [6].

Los pasos detallados del AAB se presentan en el Algoritmo 1. El algoritmo AAB se ejecuta a intervalos fijos de n ciclos en cada núcleo del procesador y consta de cuatro fases.

Medición de rendimiento de búferes. En esta fase se calcula el la cantidad de fallos por kiloinstrucción (MPKI) obtenidos con b búferes y estimados con $b+1$ búferes, así como el número de aciertos efectuados tanto en los BTAs (i.e. aciertos en bloques privados) y en la STT y los DNs (i.e. aciertos en bloques compartidos).

Filtado. En este paso se contrastan los valores obtenidos en el paso anterior con varios umbrales para determinar si el intervalo ha efectuado un número significativo de fallos y aciertos.

Análisis de Rendimiento. En esta fase se calculan dos métricas que se utilizarán para determi-

nar en el Paso 4 si el núcleo debe solicitar un nuevo búfer. La primera métrica es $MPKI_{dec\_rate}$, que indica el porcentaje de reducción de fallos por kilo-instrucción esperado al incrementar el número de búferes disponibles. En segundo lugar, $BTAs_{weight}$ indica el ratio de aciertos que el hilo o aplicación ha efectuado en su espacio privado. Esta métrica proporciona, por tanto, el *uso* que un determinado hilo está haciendo de su espacio privado.

Decisión. En esta etapa el algoritmo compara las métricas anteriores contra los umbrales Thr_{MPKI} y Thr_{weight}. Dado que los umbrales estáticos son demasiado restrictivos, en este trabajo se utilizan unos umbrales base que se escalan de acuerdo con la categoría MPKI identificada para el hilo en ejecución. Dicha categoría consiste en un número entero que actúa como multiplicador del umbral conforme al MPKI del hilo o aplicación. Si ambas métricas superan sus respectivos umbrales, se solicita un nuevo búfer al nodo BAL.

Para finalizar, obsérvese que no es estrictamente necesario ejecutar el algoritmo AAB en los tiempos exactos de finalización del intervalo. Por ello, con el objetivo de reducir la sobrecarga temporal del cómputo del algoritmo, éste puede ser ejecutado durante los ciclos de bloqueo del procesador, cuando el *Re-Order Buffer* (ROB) está bloquedo (por ejemplo, esperando un acceso a memoria de varias decenas de ciclos). De acuerdo con nuestros experimentos, el ROB está bloqueado hasta un 24% de los ciclos de procesador totales en cada intervalo (dependiendo de la aplicación). Este margen, por tanto, proporciona la suficiente flexibilidad para ejecutar el algoritmo con un mínimo impacto en las prestaciones del sistema.

IV. Marco Experimental

Los experimentos para evaluar nuestra propuesta se han realizado sobre el simulador Multi2Sim [16], cuyo código ha sido debidamente extendido para modelar FOS-Mt. Para mejorar la precisión en el modelado de latencias a memoria principal, Multi2Sim ha sido vinculado con el simulador DRAMSim2 [17]. La herramienta CACTI v6.5 [18] ha sido seleccionada para estimar el consumo de energía y la latencia de acceso de los módulos y estructuras de cache, con una tecnología base de 32nm. Finalmente, los experimentos se han realizado utilizando un subconjunto de las aplicaciones pertenecientes a las *suites* SPLASH3 2017 y ALP-Bench 2005 [19], [20].

En los experimentos, FOS-Mt se ha configurado como un banco de n búferes de tamaño k, donde cada búfer cuenta con una asociatividad de 8 vías. Para este trabajo, se ha considerado k igual a 128KB y un n igual a cuatro veces el número de cores; en total, en un multiprocesador de 8 núcleos el tamaño total del nivel FOS-Mt es de 4MB. La propuesta ha sido comparada contra los siguientes sistemas:

Private-ELC: Este sistema, que será tomado como sistema base en la mayoría de los resultados, asigna un módulo de 512KB privado a cada núcleo de ejecución y utiliza enlaces punto a punto para conectarlas con el nivel de L1.

TABLA I
Sistema base.

Core	
Núcleos	8
Frecuencia	2 GHz
Política de issue	Fuera de orden
Ancho de Issue/Commit	4 insts/ciclo
ROB	128 entradas
Jerarquía cache	
L1 I. cache	Privada, 32KB, 8 vías, 64Bytes, 2 cc
L1 D. cache	Privada, 32KB, 8 vías, 64Bytes, 2 cc
L2	Privada, 512KB, 16-vías, 64Bytes, 7 cc
Red L1-L2	
Topología	Enlace dedicado
Frequencia	2 GHz
Ancho de banda	64 Bytes/ciclo
Network L2-MC	
Topología	Malla Eléctrica
Enrutamiento	X-Y
Frequencia	2 GHz
Ancho de banda	16 Bytes/ciclo
Memoria principal & Controlador	
DRAM bus .	1600MHz
DRAM device	DDR4 (3150 Mtransfers/ciclo)
Latency	t_{RP}, t_{RCD}, t_{CL} 13.75ns
DRAM bancos	8

Shared-ELC: La configuración Shared-ELC implementa un módulo de cache monolítico de 4MB compartido por todos los núcleos. La conexión con el nivel superior (i.e. L1) se realiza mediante un crossbar eléctrico.

Shared-OPT: La configuración Shared-OPT replica la organización anterior monoítica y compartida, pero sustituye el crossbar eléctrico por un anillo óptico que ofrece las mismas prestaciones que el empleado en FOS-Mt.

NUCA-OPT: Este sistema divide la cache en n módulos de k tamaño, siendo n igual a 32 y k igual a 128KB, al igual que en FOS-Mt. Además, incorpora un anillo óptico que ofrece las mismas prestaciones que el empleado en FOS-MT. Esta configuración es, por tanto, una versión *estática* de FOS-Mt, en la que los módulos son accedidos por entrelazado de dirección y permanecen encendidos durante toda la ejecución.

Decay-OPT: Finalmente, se ha incorporado a la configuración *Shared-OPT* un mecanismo de decay que apaga las líneas de cache que no han sido accedidas durante los últimos d ciclos. Para aumentar la precisión del estudio, se han utilizado tres tamaños diferentes de d (i.e. 1, 5 y 10 millones de ciclos).

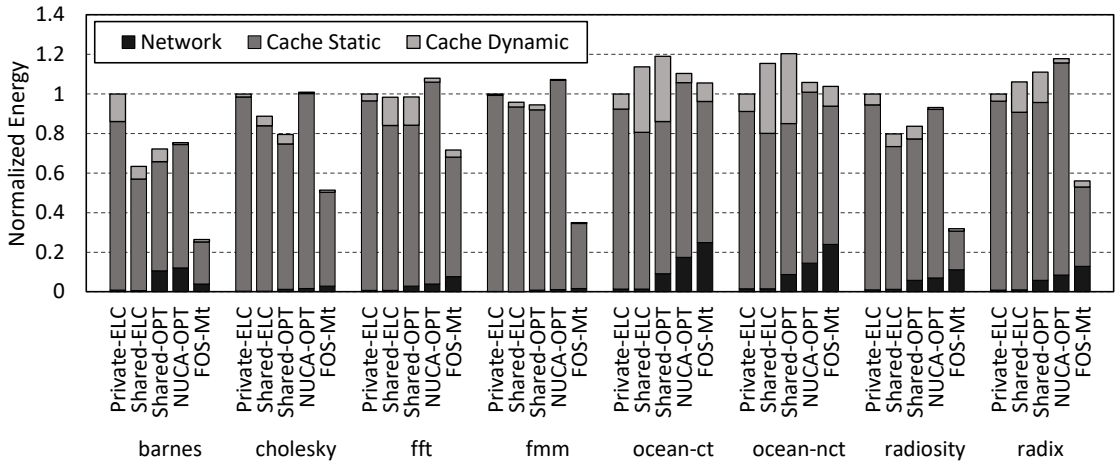


Fig. 2. Energía consumida normalizada con respecto a *Private-ELC*(I).

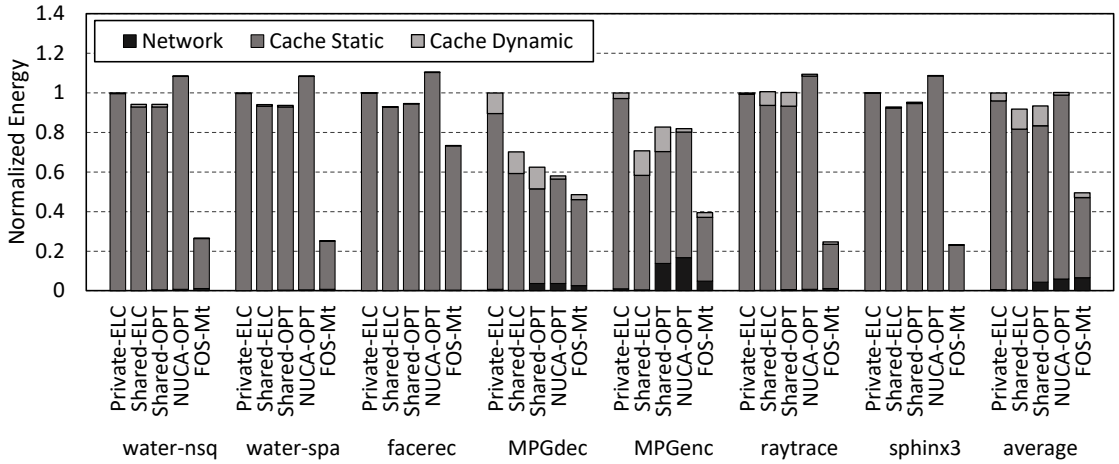


Fig. 3. Energía consumida normalizada con respecto a *Private-ELC*(II).

V. Resultados experimentales

A. Consumo de energía de FOS-Mt

Esta sección evalúa la energía consumida por FOS-Mt y la compara con los sistemas convencionales explicados en la Sección IV. Los resultados se muestran en las Figuras 2 y 3, que presentan la energía consumida normalizada con respecto a la configuración *Private-ELC*. La figura presenta la energía consumida dividida en tres componentes principales: i) la energía consumida por la red; ii) la energía estática consumida en los módulos de cache debido a corrientes de fuga; y iii) la energía dinámica consumida en cada acceso a los módulos de cache. En el caso de FOS-Mt, este último valor incluye la energía consumida en el acceso a sus estructuras dedicadas, esto es PTAs y DNs.

Los resultados obtenidos muestran como claramente la energía estática consumida en los módulos de cache es el componente que domina la energía total consumida en todas las organizaciones de cache estudiadas. Sin embargo, gracias al número de búferes que mantiene apagados durante la ejecución (i.e. un 51% de media de acuerdo con nuestros resultados), FOS-Mt consigue reducir significativamente estos valores en 10 de las 15 aplicaciones estudiadas. Las diferencias observadas en energía estática para el resto de configuraciones se deben a dos razones principales. En primer lugar, los 32 módulos de cache utilizados en el sistema *NUCA-OPT* consumen una cantidad mayor de energía que el módulo monolítico de 4MB empleado en las configuraciones compartidas *Shared-OPT* y *Shared-ELC*. La segunda razón atiende al tiempo de ejecución conseguido por cada uno de los enfoques estudiados. Por ejemplo, en el caso de `barnes`, el tiempo de ejecución bajo la configuración *Private-ELC* es un 40% mayor que en el resto de configuraciones, lo que se traduce en un aumento de la energía total consumida.

En lo que respecta al consumo de energía dinámica, se puede observar que *NUCA-OPT* es la configuración que presenta los mejores resultados, seguida de FOS-Mt. Esto se debe a la baja complejidad de los módulos de 128KB utilizados en estas configuraciones, ya que gracias a ésta se reduce el coste energético por acceso. Del mismo modo, los grandes módulos monolíticos empleados en las configuraciones *Shared-OPT* y *Shared-ELC* presentan las cifras más altas de consumo de energía dinámica. En el caso de FOS-Mt, el ligero incremento sobre *NUCA-OPT* obedece a la energía consumida por las estructuras hardware especiales de este sistema, que

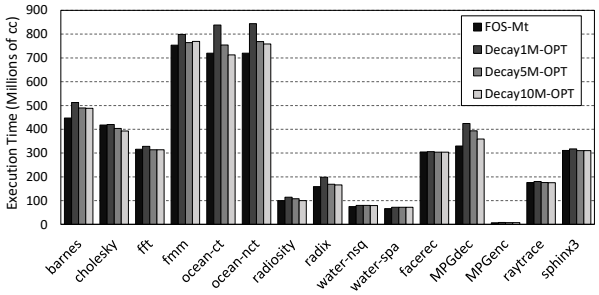


Fig. 4. Tiempo de ejecución de FOS-Mt y los distintos sistemas de Cache Decay.

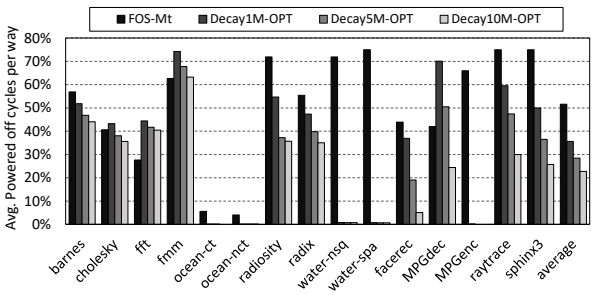


Fig. 5. Porcentaje medio de ciclos apagados por vía durante la ejecución.

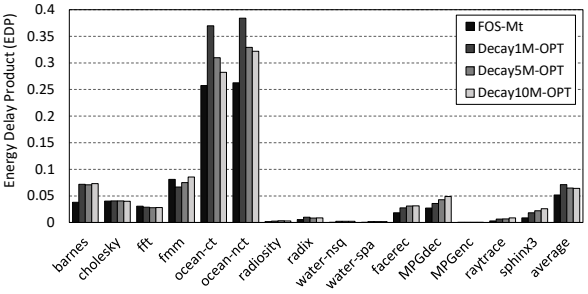


Fig. 6. EDP de FOS-Mt y los distintos sistemas de Cache Decay.

representa de media un 4% adicional sobre el total de energía consumida.

Finalmente, atendiendo al consumo energético proveniente de la red de interconexión, se aprecia que la energía consumida por las configuraciones con conexión óptica es de 8 a 12 veces mayor que la observada en las configuraciones eléctricas *Private-ELC* y *Shared-ELC*. Esto supone un incremento de hasta un 10% de la energía total consumida en los sistemas con interconexión óptica. Sin embargo, obsérvese que la red de interconexión óptica no es un componente que forma parte del diseño de FOS-Mt, ya que este sistema puede ser implementado con cualquier red que satisfaga los requisitos de latencias reducidas y con baja variabilidad. Por ejemplo, en el supuesto de que el sistema FOS-Mt fuera implementado con una red eléctrica de altas prestaciones que garantizara ambos requisitos, el ahorro de energía conseguido por el algoritmo AAB se mantendría.

B. Comparativa entre FOS-Mt y Cache Decay

En la sección anterior se ha explicado la reducción de energía conseguida por FOS-Mt con respecto a cuatro sistemas convencionales. Sin embargo, ninguno de estos sistemas está específicamente diseñado para ahorrar energía, como sí lo está nuestra propuesta. Por ello, en esta sección FOS-Mt es comparado contra un sistema que implementa la técnica bien conocida de ahorro de energía *Cache Decay*. Dado que el tiempo de vida de un bloque de cache almacenado en el nivel L2 o inferior puede presentar gran variabilidad entre aplicaciones, se han estudiado tres configuraciones de decay diferentes variando el tiempo de intervalo de decay. Las tres longitudes de intervalo estudiadas han sido 1, 5 y 10 millones de ciclos. Estos números se utilizan en la presentación de resultados para identificar el correspondiente esquema de decay. Por ejemplo, en la configuración *Decay1M-OPT* una línea de cache que no ha sido accedida durante el último millón de ciclos de procesador es invalidada y apagada.

La Figura 4 muestra el tiempo de ejecución (en millones de ciclos de reloj) alcanzado por las configuraciones de decay estudiadas y por FOS-Mt. Como se observa, la reducción del intervalo de decay se traduce directamente en un incremento en el tiempo de ejecución. Por lo tanto, cuanto mayor es el intervalo de decay, mejores son las prestaciones del sistema; por ello, el mejor sistema de decay de los estudiados desde el punto de vista de las prestaciones es *Decay10M-OPT*. Comparado con esta configuración, FOS-Mt mejora o iguala las prestaciones para todas las cargas excepto para el caso de `cholesky`.

Aunque incrementar el intervalo de decay ayuda a mejorar las prestaciones del sistema, longitudes de intervalo demasiado grandes implican una reducción en el tiempo medio que una línea de cache permanece apagada durante la ejecución. Este efecto adverso desde el punto de vista del consumo energético se muestra en la Figura 5, que presenta el porcentaje medio de tiempo que las líneas de cache han permanecido apagadas para cada aplicación. Como se observa, este porcentaje es de un 51% de media en el caso de FOS-Mt, mientras que en el caso de *Decay1M-OPT* (el mejor intervalo desde el punto de vista energético) el porcentaje decae hasta un 37%.

Finalmente, la Figura 6 muestra en *Energy Delay Product* (EDP) de las configuraciones de decay estudiadas así como de FOS-Mt. Tal y como se aprecia en la figura, FOS-Mt obtiene los mejores valores de EDP en 13 de las 15 aplicaciones estudiadas, con las excepciones de `ftt` y `fmm`. En lo que respecta a las variantes de decay, tanto *Decay5M-OPT* como *Decay10M-OPT* obtienen resultados similares, mientras que *Decay1M-OPT* es la configuración que presenta el mayor (i.e. el peor) EDP entre todas las configuraciones estudiadas. Estos resultados coinciden con lo esperado ya que obedecen al bajo rendimiento en términos de tiempo de ejecución observado para *Decay1M-OPT*.

En conclusión, estos resultados demuestran que FOS-Mt no sólo consigue mayor ahorro de energía de media que los sistemas de decay estudiados, sino

que también presenta el tiempo de ejecución más reducido para la mayoría de las aplicaciones ejecutadas. Además, tal y como se menciona en la introducción del presente trabajo, cabe recordar que la implementación práctica de sistemas que abordan la gestión de la energía a la granularidad de la línea de cache presenta grandes dificultades.

VI. Conclusiones

En este trabajo se ha introducido FOS-Mt, una organización de cache para multiprocesadores de bajo consumo que ofrece un balance entre rendimiento y consumo de energía. La arquitectura propuesta cuenta con un espacio de cache de dos niveles, cuyo segundo nivel es único, compartido por todos los núcleos y dividido en múltiples y pequeños búferes de memoria cache. Estos búferes son asignados dinámicamente a los núcleos de ejecución cuando se espera obtener una mejora de rendimiento por ello. Gracias a que los búferes son mucho mayores que las líneas de cache individuales, la implementación práctica de estrategias de ahorro de energía en este trabajo es mucho más fácil con respecto a propuestas anteriores.

Los resultados obtenidos muestran que FOS-Mt reduce hasta un 78% la energía estática consumida debido a corrientes de fuga en los módulos de cache. Además, gracias al reducido tamaño de los búfres, la energía dinámica también se reduce con respecto a organizaciones de cache convencionales. Finalmente, aunque el enfoque bien conocido *Cache Decay* presenta serias restricciones de implementación, también se ha comparado FOS-Mt contra esta propuesta. De acuerdo con los resultados experimentales, FOS-Mt obtiene una mejora en el EDP de un 19.3% sobre el mejor intervalo de decay, de media para todas las cargas ejecutadas. En algunos casos, esta mejora es de hasta un 48%.

Agradecimientos

Este trabajo ha sido financiado por los fondos del Ministerio de Economía y Competitividad (MINECO) bajo la subvención RTI2018-098156-B-C51.

Referencias

[1] Stefanos Kaxiras, Zhigang Hu, and Margaret Martonosi, "Cache decay: Exploiting generational behavior to reduce cache leakage power," in *Procs. of the 28th Annual International Symposium on Computer Architecture*, 2001, ISCA'01, pp. 240–251.

[2] K. Flautner, Nam Sung Kim, S. Martin, D. Blaauw, and T. Mudge, "Drowsy caches: simple techniques for reducing leakage power," in *Proceedings 29th Annual International Symposium on Computer Architecture*, May 2002, pp. 148–157.

[3] Sparsh Mittal, "A survey of architectural techniques for improving cache power efficiency," *Sustainable Computing: Informatics and Systems*, vol. 4, no. 1, pp. 33 – 43, 2014.

[4] R. Ubal, J. Sahuquillo, S. Petit, H. Hassan, and P. Lopez, "Leakage current reduction in data caches on embedded systems," in *The 2007 International Conference on Intelligent Pervasive Computing (IPC 2007)*, Oct 2007, pp. 45–50.

[5] Michael Powell, Se-Hyun Yang, Babak Falsafi, Kaushik Roy, and T. N. Vijaykumar, "Gated-vdd: A circuit technique to reduce leakage in deep-submicron cache memories," in *Proceedings of the 2000 International Symposium on Low Power Electronics and Design*, New York, NY, USA, 2000, ISLPED '00, pp. 90–95, ACM.

[6] M.K. Qureshi and Y.N. Patt, "Utility-based cache partitioning: A low-overhead, high-performance, runtime mechanism to partition shared caches," in *MICRO*, Dec 2006, pp. 423–432.

[7] Seongbeom Kim, Dhruba Chandra, and D. Solihin, "Fair cache sharing and partitioning in a chip multiprocessor architecture," in *PACT*, Sept 2004, pp. 111–122.

[8] Jichuan Chang and Gurindar S. Sohi, "Cooperative caching for chip multiprocessors," in *Procs. 33rd Annual Int. Symp. on Computer Arch.*, 2006, pp. 264–276.

[9] Enric Herrero, José González, and Ramon Canal, "Distributed cooperative caching," in *Procs. of the 17th International Conference on Parallel Architectures and Compilation Techniques*, 2008, PACT '08, pp. 134–143.

[10] Enric Herrero, José González, and Ramon Canal, "Elastic cooperative caching: An autonomous dynamically adaptive memory hierarchy for chip multiprocessors," in *Procs. of the 37th Annual International Symposium on Computer Architecture*, 2010, ISCA '10, pp. 419–428.

[11] M. Awasthi, K. Sudan, R. Balasubramonian, and J. Carter, "Dynamic hardware-assisted software-controlled page placement to manage capacity allocation and sharing within large caches," in *2009 IEEE 15th International Symposium on High Performance Computer Architecture*, Feb 2009, pp. 250–261.

[12] Bradford M. Beckmann, Michael R. Marty, and David A. Wood, "Asr: Adaptive selective replication for cmp caches," in *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, Washington, DC, USA, 2006, MICRO 39, pp. 443–454, IEEE Computer Society.

[13] S. Cho and L. Jin, "Managing distributed, shared l2 caches through os-level page allocation," in *2006 39th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'06)*, Dec 2006, pp. 455–468.

[14] Zeshan Chishti, Michael D. Powell, and T. N. Vijaykumar, "Optimizing replication, communication, and capacity allocation in cmps," *SIGARCH Comput. Archit. News*, vol. 33, no. 2, pp. 357–368, May 2005.

[15] Jaehyuk Huh, Changkyu Kim, Hazim Shafi, Lixin Zhang, Doug Burger, and Stephen W. Keckler, "A nuca substrate for flexible cmp cache sharing," in *Procs. of the 19th Annual International Conference on Supercomputing*. 2005, ICS '05, pp. 31–40, ACM.

[16] R. Ubal, J. Sahuquillo, S. Petit, and P. Lopez, "Multi2sim: A simulation framework to evaluate multicore-multithreaded processors," in *Int. Symp. on Computer Architecture and High Performance Computing.*, pp. 62–68.

[17] Paul Rosenfeld, Elliott Cooper-Balis, and Bruce Jacob, "Dramsim2: A cycle accurate memory system simulator," *IEEE Comput. Archit. Lett.*, pp. 16–19.

[18] Naveen Muralimanohar, Rajeev Balasubramonian, and Norman P. Jouppi, "Cacti 6.0: A tool to model large caches," in *HP Laboratories*, 2009.

[19] C. Sakalis, C. Leonardsson, S. Kaxiras, and A. Ros, "Splash-3: A properly synchronized benchmark suite for contemporary research," in *2016 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, April 2016, pp. 101–111.

[20] Man-Lap Li, R. Sasanka, S. V. Adve, Yen-Kuang Chen, and E. Debes, "The alpbench benchmark suite for complex multimedia applications," in *Proceedings of the IEEE International Workload Characterization Symposium, 2005*, 2005, IIWC'05.

Docencia en Arquitectura y Tecnología de Computadores

Saliendo del bucle

Francisco Fernández de Vega
Universidad de Extremadura
fcofdez@unex.es

Resumen—**Este artículo trata sobre la oportunidad que la programación funcional nos ofrece en los procesos de enseñanza para que los estudiantes sean conscientes de las dependencias de datos y sus implicaciones en los modelos de programación paralela y distribuida, fundamental en el siglo XXI. Aunque son otras las metodologías que actualmente gozan de más difusión, tal como la Programación Orientada a Objeto (OOP), el problema es que los modelos secuenciales son inherentes a las mismas, y una vez que los estudiantes han entrado en el marco de trabajo, les cuesta desembarazarse de esta "camisa de fuerza" para entrar de lleno y sacar provecho de las arquitecturas paralelas. En este trabajo se presenta la Programación Funcional Sin Bucles, modelo que replica la prohibición en el pasado de las sentencias "go-to" en la programación estructurada aplicada ahora a las estructuras de bucle. Tal como se describe en este trabajo, el modelo, ha sido aplicado con éxito a jóvenes estudiantes de secundaria, que han desarrollado un pensamiento abstracto para resolver problemas de programación paralelo de forma natural.**

Palabras clave—**Programación funcional; programación paralela; recursividad.**

I. Introducción

Aunque la programación funcional no ha sido la más popular en ciencias de la computación durante décadas [4], son varios los estudios que muestran la importancia de este paradigma y sus ventajas sobre otros más favorecidos por las modas, tal como la OOP lo ha sido en los últimos veinte años, o como la programación procedural (estructurada) lo fue durante los años ochenta y comienzo de los noventa. En cualquier caso, no sólo la industria ha vuelto la espalda a la programación funcional, también la academia suele olvidar este paradigma. Desafortunadamente, los paradigmas influyen notablemente en el desarrollo del pensamiento abstracto, de modo que los estudiantes se acostumbran a resolver los problemas de un modo influido directamente por el paradigma.

En este artículo estamos interesados en la manera en que los estudiantes adquieren algunos de los principios de programación más importantes y que se relacionan con el modo en que tareas repetitivas sobre datos, y datos con dependencias, puedan resolverse de manera secuencial o paralela.

Son varias las características que distinguen la programación funcional de sus alternativas: programación declarativa, funciones sin estado, o recursividad como base de repetición de tareas, por nombrar sólo algunas. La última en particular, es de interés primordial como veremos, y, aunque la recursividad está presente en todos los paradigmas y lenguajes de programación, por razones que a continuación discutimos, los estudiantes huyen de este concepto y prefieren aplicar soluciones alternativas basadas en bucles.

La programación funcional tiene un apoyo notable en la recursividad, herramienta esta última basada en la recurrencia de tareas repetitivas. Desafortunadamente, la recursividad, presente en otras metodologías, ha sido analizada como uno de los conceptos más difíciles de aprender en programación recursiva [1]. Los estudiantes generalmente prefieren soluciones directas basadas en bucles en lugar de lo que consideran retorcidas maneras de resolver recursivamente los problemas. Pero puede que una de las razones de este comportamiento, más allá de la posible dificultad del concepto, esté en que los profesores de hoy también fueron antiguos estudiantes que rehuían la recursividad en favor de los bucles.

Hay quienes cuestionan la idoneidad de enseñar en primer lugar lenguajes del ámbito OOP. Incluso hay quienes propone enseñar primero el modelo funcional [2], y la recursividad como concepto primordial, para después pasar a los bucles, siempre que hay un profesor competente a cargo de la enseñanza [3]. Pero nosotros pondremos el foco en algo que hasta dónde sabemos, solo muy recientemente se ha discutido [13], y que se contextualiza en los procesos actuales de "Big Data", que requieren la repetición masiva de tareas idénticas. Esta idea ha sido la base de los actuales modelos *map/reduce*: mientras que una operación "map" aplica una misma operación a múltiples datos independientes, la operación "reduce", que consideramos aquí una versión simplificada de un proceso recursivo, aplica un proceso secuencial a datos dependientes. Utilizando este par de funciones, los programadores pueden aplicar tareas paralelas a datos, ya sean con dependencias o sin ellos, si necesidad de utilizar bucles. Y es este precisamente el punto que queremos discutir aquí: una enseñanza adecuada de la programación funcional con prohibición expresa de uso de bucles, permitiría a los estudiantes desarrollar un modelo natural de programación paralela mucho más productivo, del mismo modo que la prohibición de las sentencias "go-to" mejoró notablemente la legibilidad, mantenibilidad y en definitiva la productividad de los programadores software en la era de la programación estructurada.

No obstante, la prohibición de los bucles no es suficiente. Un uso indiscriminado de estructuras recursiva provocaría el mismo problema del que huimos: código secuencial producido de forma irreflexiva por los programadores.

Este trabajo analiza la forma adecuada de utilizar la programación funcional como mejor metodología para

la comprensión de las dependencias de datos y el desarrollo de un modelo de pensamiento paralelo automático a la hora de producir código, que podrá así aprovechar los recursos de cómputo paralelos disponibles en la actualidad en cualquier entorno.

Aunque la adopción generalizada del modelo aquí propuesto llevaría a los futuros programadores a prescindir de los bucles, esto no dificulta que como concepto puedan también aprenderlo y conocerlo para realizar tareas de mantenimiento de código "antiguo", del mismo modo que hoy el lenguaje COBOL no forma parte de los planes docentes, pero aún hay programadores que lo aprenden posteriormente y utilizan para mantener aplicaciones bancarias de otra época.

El resto del trabajo se estructura del siguiente modo: En la sección II describimos algunas ventajas de la programación funcional, y se analiza la situación actual de la enseñanza de la programación. La sección III analiza las tareas repetitivas desde el punto de vista del bucle y su problemática. La sección IV describe nuestra propuesta y la sección V los experimentos y resultados obtenidos. Por último, la sección VI presenta las conclusiones.

II. La naturaleza recursiva de la programación funcional.

Como hemos descrito anteriormente, la programación funcional se caracteriza, entre otros aspectos, por lo siguiente:

1. Funciones de alto orden: funciones que utilizan otras funciones como parámetros.
2. Funciones puras: Sin efectos laterales, sin estado interno.
3. Recursividad: Uso intensivo de funciones que se invocan a si mismas para resolver procesos iterativos secuenciales.
4. Macros: Los datos pueden transformarse en código y ser ejecutado en tiempo de ejecución.

Algunas de estas características son compartidas con otras metodologías de programación, pero las posibilidades que ofrecen todas ellas a la vez son únicas en programación funcional. Aunque todas son importantes, nos centraremos aquí en las funciones de alto orden y en la recursividad, porque pueden jugar un rol fundamental en los procesos de aprendizaje de los estudiantes al enfrentarse a la programación paralela.

La PF tiene sus orígenes en el "cálculo lambda" desarrollado por Alonzo Church [6], un modelo de computación universal que se basa en la aplicación sucesiva de funciones: funciones que aplican otras funciones a resultados generados por otras funciones, y así sucesivamente. Tal como mostraremos más adelante, esta posibilidad, utilizada adecuadamente, permite describir e implementar operaciones paralelas en el paradigma funcional, en un modelo conocido como programación "declarativa", frente a la "imperativa" tradicional. Por otro lado, la *recursividad* también permite describir operaciones que se realizan múltiples veces, aunque en esta ocasión mediante una secuencia que se desarrolla a lo largo del tiempo. Estas diferencias importantes, pueden ser utilizadas provechosamente para entender las *dependencias* que los datos pueden presentar y que afecta al modelo paralelo/secuencial de procesamiento.

En las metodologías actuales de enseñanza de la programación, los *bucles* son la herramienta fundamental para resolver tareas repetitivas. Así, cuando actualmente en la OOP los estudiantes resuelven procesos repetitivos, los bucles son su mecanismo principal para asegurar que una tarea se realiza un número determinado de veces. Aunque las funciones recursivas también podrían ser aplicadas, esta técnica sólo se muestra al alumno una vez que domina el bucle. Más aún, la recursividad se enseña a los estudiantes para ser aplicada en problemas en los que la recursividad es el mecanismo *natural* para resolver un problema (por ejemplo, la serie de Fibonacci), y no se plantea su uso generalizado para resolver iteraciones. Igualmente, su uso es aplicado sobre ciertas estructuras de datos, por ejemplo. cuando se trabaja con operaciones tipo "recorrido de un árbol", y, sobre todo, porque la solución equivalente mediante bucles requeriría un manejo adecuado de una estructura de datos tipo pila, que sería para los estudiantes más complicado que la propia recursividad.

En definitiva, la enseñanza basada en bucles es probablemente la razón por la que los estudiantes (y futuros profesionales) evitan la recursividad tan frecuentemente como les es posible [7]. Son varios los estudios previos que muestran la recursividad como un concepto difícil de aprender para los programadores nobeles [5]

A. La situación en Extremadura (y más allá)

Aunque incluimos a continuación datos obtenidos en Extremadura, entendemos que la situación a nivel nacional es similar, dado que las metodologías OOP dominan el panorama.

En nuestra universidad, un reciente concurso de programación, abierto a cualquier interesado, *El reto Ada Byron*, permitió la participación de 40 estudiantes de diferentes niveles educativos (universidad, bachillerato, ciclos formativos). Después de revisar las soluciones a cada uno de los problemas planteados, constatamos que nadie utilizó procedimientos recursivos (ni lenguajes funcionales).

Como segundo ejemplo de la situación, nos fijamos en la asignatura Sistemas Operativos, obligatoria en el tercer año de Ingeniería Informática en Sistemas de Información. De los treinta estudiantes matriculados en el curso actual, sólo uno de ellos utilizó una solución recursiva para alguno de los problemas planteados.

Por último, una reciente encuesta realizada en una empresa de desarrollo de software, *software Factory,* ubicada en Extremadura, mostró que el 75% de quienes completaron el formulario (53) no había utilizado la recursividad en el último año. Ante la pregunta de la

razón para ello, indicaron mayoritariamente, que *no es necesaria la recursividad o todo puede hacerse más fácilmente con bucles.*

Pero además de esto, planteamos la pregunta sobre el uso que hacen de la programación paralela, hoy que disponemos recursos de cómputo paralelo en todos los dispositivos que utilizamos: incluyendo teléfonos con varios núcleos de procesamiento. De igual modo, un 75% nos indicaron que no habían hecho uso de la misma en el último año.

Aunque son elementos aparentemente alejados, la programación paralela y las técnicas de programación recursiva, creemos que hay una línea fundamental que las conecta, y que desgraciadamente las metodologías actuales de programación olvidan y provocan lo que se manifiesta en los datos anteriores. Nuestro resumen del diagnóstico es el siguiente: (i) cuando los estudiantes aprenden programación por primera vez, la recursividad se evita, y se utilizan los bucles para procesos iterativos; (ii) cuando el estudiante llega a la recursividad, el bucle es tan asumido, que aunque se aprenda el concepto, el estudiante y futuro profesional tenderá a evitarlo; (iii) la programación secuencial es el modelo aplicado mayoritariamente por estudiantes y profesionales, aunque en los últimos años de los estudios tienen oportunidades de aprender programación concurrente, paralela y/o distribuida.

Destacamos como dato último y que corrobora todo lo expuesto, que en el Centro Universitario de Mérida no se ha presentado ni un solo Trabajo Fin de Grado que utilice programación paralela en los último 5 años.

Creemos por todo lo anterior, que los *bucles* son el origen y causa de los problemas detectados, y pensamos, que del mismo modo que los *saltos* fueron suprimidos en su día de las metodologías de programación, deberíamos considerar en la actualidad *prohibir los bucles* en las metodologías actuales de programación, si queremos que los profesionales saquen provecho de las arquitecturas multi-core y many-core actualmente disponibles.

Describimos a continuación porqué pensamos que este sería el camino hacia una mejor enseñanza de la programación que el siglo XXI requiere: programación paralela y distribuida.

III. SER O NO SER

Tal como decíamos anteriormente, una vez los estudiantes aprenden las técnicas de programación secuencial y entienden las estructuras basadas en bucles, aplicarán sistemáticamente lo aprendido en cualquier proceso repetitivo que se encuentren: cuando dos opciones están disponibles, la más fácil es siempre la elegida (bucles frente a recursividad). Esto sucede independientemente de la metodología que se utilice. De hecho, cuando los estudiantes aprenden su primer lenguaje funcional después de haber pasado por OOP, una de sus primeras preguntas se refiere a cómo construir bucles en el nuevo lenguaje. Veremos a continuación que la programación funcional hace hincapié en el uso de estructuras alternativas a los bucles, y que esta forma de trabajo permite un pensamiento abstracto diferente cuando se encaran tareas repetitivas, lo que es fundamental en contextos de programación paralela.

Sin embargo, no es suficiente con prohibir los bucles en programación funcional. Este método simplista, si no es convenientemente acompañado con otras estrategias, produciría el mismo problema que denunciamos en este trabajo sobre el uso abusivo de los bucles: podría provocar resolver todas las tareas repetitivas con procesos recursivos, y de nuevo quedaríamos encerrados en métodos de programación secuencial, que no es el objetivo.

A. Ser o no ser, esa es la cuestión.

Para entender lo anterior, utilizaremos como metáfora la enseñanza de una lengua a hablantes no-nativos; concretamente nos centraremos en el español como segundo idioma para los anglo-parlantes.

Cuando un estudiante de español aprende a utilizar los verbos “ser” y “estar”, y entiende su equivalencia con el “to be” en inglés, sufre sus primeros errores gramaticales. Aunque diferentes en español, equivalen a un único verbo en inglés. Así, cuando construyen la famosa sentencia “To be or not tobe, that is the question”, por error pueden construir en español: “Estar o no estar, esa es la cuestión”, cuyo significado es completamente diferente a la correcta “ser o no ser, esa es la cuestión”. Para los estudiantes de español, no es fácil la elección del verbo al principio.

B. Map o reduce, esa es la cuestión recursiva.

De forma similar a lo que ocurre con los nativos ingleses estudiantes de español, creemos que sucede con los “nativos” del bucle, que encuentran dificultades para entender una dimensión muy importante de los datos que van a ser procesados de forma repetitiva: las dependencias, y cómo esta afectan al modo en que un problema debería ser correctamente resuelto cuando hay recursos de computación paralelo disponibles. Los estudiantes que se han formado bajo el paraguas de los bucles, han desarrollado la tendencia de elegir siempre estructuras de bucle sin considerar previamente la relación que los datos tienen entre sí, y por eso no se plantean y les cuesta aprender la programación paralela por la misma razón, si el problema puede resolverse más eficientemente de algún otro modo.

Para entender mejor lo anterior partamos del siguiente par de ejemplos que ilustra la cuestión:

- a[i]=a[i]+1; //añadir uno a cada posición del vector.
- a[i]=a[i]+a[i-1]; //acumular secuencialmente las posiciones del vector utilizando las de la posición anterior y la propia.

Como podemos observar, para el primer problema, cada valor podría ser calculado de forma independiente, lo que significa que todos los valores se podrían obtener simultáneamente (paralelo). Sin embargo, en el segundo caso, es necesario un orden estricto de las operaciones para obtener los valores correctos, y por tanto las

operaciones deben realizarse en una secuencia estricta (secuencial).

A pesar de estas diferencias notables, ambos problemas serán resueltos del mismo modo en un contexto basado en bucles, y los estudiantes producirán soluciones como las siguientes:

```
for (i=0;i<n;i++)
  a[i]++;
```

```
for (i=1;i<n;i++)
   a[i]=a[i]+a[i-1];
```

Aunque ambas soluciones son válidas cuando se ejecutan de forma secuencial, si paralelizamos los bucles, las cosas no serán tan sencillas en el segundo caso, debido precisamente al orden en que los "n" posibles procesos deberían trabajar.

Si nos fijamos bien en las dos soluciones, y aunque desde el punto de vista de lo que el programador ha hecho son idénticas, ya que se trata de visitar las posiciones de un vector en orden creciente, hay una gran diferencia que para los estudiantes puede pasar desapercibida: mientras que en el primer problema el orden en que se realicen no afecta al resultado –independencia de datos- en el segundo sí que afecta –dependencia de datos. Así, a la hora de construir una solución paralela, este hecho es fundamental, pero, y debido al modelo de aprendizaje descrito, el estudiante no es consciente de las diferencias y simplemente tratará de crear n procesos diferentes cada uno realizando una operación simple sobre una posición del vector.

```
a[i]++;     // primer problema

a[i]=a[i]+a[i-1];// seg. problema
```

Obviamente, el segundo problema no funcionará bien debido a la velocidad relativa de los procesos.

Podríamos analizar estos dos ejemplos desde el punto de vista de *la taxonomía de Flynn:* son parte de la clase SIMD, una única operación (suma) a aplicar a muchos datos (cada posición del vector). En el primer problema, si la versión paralela es lanzada, dado que cada proceso trabaja con un único elemento no habría dificultad alguna en obtener los resultados; pero en el segundo caso, dependiendo de la velocidad relativa de los procesos, diferentes ejecuciones darían lugar a diferentes resultados.

Podemos resumir todo lo anterior del siguiente modo: (i) estudiantes y profesionales aplican bucles para resolver procesos repetitivos; (ii) los bucles ocultan las dependencias de datos; (iii) la programación estructurada y OOP, debido a los dos puntos anteriores, ocultan a los estudiantes la naturaleza de los datos, y les hace difícil entender y aplicar la programación paralela.

Pero tenemos oportunidades para hacer a los estudiantes más conscientes de las dependencias de datos, si nos fijamos en las funciones de alto orden de la programación funcional. Más aún cuando las soluciones basadas en estas funciones de alto orden son tan sencillas que se convierten en el estándar de facto para los procesos repetitivos, y sólo son sustituidas por las operaciones recursivas cuando la dependencia de datos nos obliga a ello.

En la siguiente sección mostramos con detalle la metodología propuesta.

IV. METODOLOGÍA: PROHIBIENDO LOS BUCLES.

La propuesta que describimos se basa en la prohibición del bucle en los lenguajes de programación, de modo similar a como los saltos fueron prohibidos en la programación estructurada en los años ochenta, y desde entonces no se utilizan explícitamente en los lenguajes de alto nivel.

Esta propuesta podrá llevarse a cabo en lenguajes que incorporan el paradigma funcional, en los que tanto la recursión como las funciones map (funciones que aplican funciones) están disponibles. Si esto último no existe en el lenguaje elegido, la metodología propuesta no podrá aplicarse. Para simplificar, en lo que sigue nos referiremos a la programación funcional como nuestro marco de trabajo, y mostraremos ejemplos escritos en el lenguaje *common lisp*.

Volviendo a los ejemplos mostrados anteriormente, y teniendo en cuenta el uso de funciones map y la recursividad (que puede verse como una función reduce en el paradigma funcional), y sabiendo que los bucles ya no son una opción para los estudiantes, la solución al primer problema sería del tipo (siendo *a* la lista –vector en el modelo previo- con los valores):

```
(mapcar '1+ a)
```

Este mismo problema, primer problema del ejemplo, podría también resolverse recursivamente del siguiente modo:

```
(defun solution(a)
   (if (null a)
         a
     (cons (1+ (car a))
                (solution (cdr a)))))
```

y de forma equivalente mediante una función *reduce*:

```
(reduce (lambda (x y)
   (append x
```

```
    (list (1+ y))))
      (cons (list (car a))(cdr a)))
```

Aunque la función *reduce* es equivalente a las funciones recursivas, recomendamos desde aquí no enseñarlas hasta que se domina la recursividad y las funciones lambda.

Si nos fijamos en la solución basada en *mapcar* y en la recursiva, entenderemos fácilmente que el estudiante y profesional de la programación prefiere en igualdad de condiciones elegir la versión *mapcar*. Y esto es precisamente lo que buscamos, que la solución *map* sea la preferida siempre que sea posible, dado que es una función intrínsecamente paralela y, podría lanzar la ejecución de cada tarea (operación sobre el vector en este caso) en hilos o procesos diferentes. Pero el estudiante novato ni siquiera es consciente de la posibilidad de la ejecución paralela. El simplemente aprende a programar con las herramientas que se le dan.

Ahora bien, si consideramos de nuevo el segundo problema con la metodología descrita, y asumiendo que el estudiante trata de construir una solución *map* por defecto, probablemente construya en su primer intento un código similar al que mostramos a continuación:

```
(mapcar '+ (cons '0 a) (append a '(0)))
```

Esta solución podemos apreciar que no funciona: si la lista de entrada contiene la serie de números 1, 2, 3… nos devolverá una lista formada por 1, 3, 5… en lugar del resultado correcto 1, 3, 6. Esto sucede por el modo paralelo intrínseco de trabajo de las funciones map. Incluso si el profesor no explica las especiales características del problema, y las dependencias de datos, el estudiante comprobará que es imposible resolverlo con funciones *map*, y, a pesar de su tendencia a rehuir los procesos recursivos, entenderá que la única forma de resolverlo es como una secuencia de pasos llevados a cabo mediante recursividad. Es decir, la propia metodología de programación sin bucles le llevará a construir soluciones como la siguiente:

```
(defun solution (a)
  (if (null (cdr a))

      a

    (cons (car a)
       (solution

            (cons (+ (car a)

                     (cadr a))
                  (cddr a))))))
```

o su equivalente basado en *reduce*:

```
(reduce (lambda (x y)
  (append x

          (list (+ (car (last x)) y))))
  (cons (list (car a)) (cdr a)))
```

Por todo lo anterior, una vez que el estudiante descubre la naturaleza fundamental de las funciones map y reduce (procesos recursivos), su forma de trabajar (paralelo/secuencial) y sus implicaciones en las dependencias de datos, el estudiante simplemente escribirá código con la metodología que ha aprendido sea cual sea el problema que le planteen, y el código estará listo para ser ejecutado en entornos paralelos. Así, el paradigma funcional desprovisto de bucles hace conscientes a los estudiantes sobre la naturaleza de los datos, y la forma correcta de procesarlos, y permite que el código que escribe pueda ejecutarse en secuencia o en paralelo sin cambios (cambios despreciables si se compara con otras metodologías: inclusión de librería paralela, y cambio de función "map" por "pmap"; eso es todo). Obviamente, no estamos aquí considerando otras muchas cuestiones de la programación paralela, que requieren el uso de mecanismos para control de bloqueos entre procesos. Tratamos sólo el procesamiento masivo de datos a los que se aplica una tarea idéntica, objeto de los modelos map/reduce cada vez más extendidos.

El aspecto clave de la metodología descrita es que el estudiante desarrolla un modo de pensar en la programación intrínsecamente *paralelo*, y cuando lo domina, puede aplicarlo y entender más fácilmente otras metodologías de programación paralelas y otros conceptos más avanzados de la misma. Pensamos que este modelo puede y debe ser aplicado desde el principio, como primera metodología, incluso con jóvenes que se adentran en la programación antes de llegar a la universidad; su aprendizaje no tiene porqué ser más difícil del tradicional basado en bucles.

En la siguiente sección describimos la experiencia desarrollada al aplicar la metodología a un grupo de estudiantes de secundaria.

V. RESULTADOS

Por todo lo anterior, decidimos enseñar programación funcional como primera metodología a un grupo de estudiantes de secundaria, que participaban en el programa STEM Escuelas Municipales de Jóvenes Científicos en Extremadura (MSYS)[10], que durante todo el curso escolar realiza talleres semanales con más de 400 estudiantes en 20 localidades extremeñas.

Un pequeño grupo de estudiantes cuyo principal interés es la programación comenzaron a trabajar con el modelo arriba descrito. Aprendieron así a utilizar tanto la función *map* como los procedimientos *recursivos*, y no se les habló ni se les mostró el concepto de bucle, aunque todo hay que decirlo, lo conocían de sus clases de tecnología en el instituto, dónde habían practicado con *scratch*. La idea era comprobar si el nuevo modelo permite a los jóvenes estudiantes entender desde el

principio los problemas con las dependencias de datos. El lenguaje de programación elegido fue Common Lisp, utilizando la máquina virtual CLISP [12].

Después de cuarenta horas de enseñanza práctica, se les propuso un par de problemas para ver si habían entendido adecuadamente dos ideas fundamentales: (i) el uso de funciones recursivas para resolver tareas que se repiten en una secuencia de pasos ordenada; (ii) que las tareas repetitivas sin dependencias de datos se deben y pueden resolver con *map*, en un proceso de naturaleza paralelo.

Los dos problemas propuestos fueron los siguientes:

1) Problema sin dependencias (paralelo): Calcular una serie de volúmenes de cilindros: Dados los *radios* y *alturas* de una serie de cilindros, el estudiante debe calcular los volúmenes de los mismos. La serie de radios y Alturas se suministra en un par de listas, una para cada serie de datos (r-list y h-list). No valoramos aquí la conveniencia de suministrar los datos de esta manera, o de cualquier otra. Lo que interesa es el procedimiento decidido por el estudiante para resolver el problema. Puede observarse que el problema puede por naturaleza resolverse en paralelo, dado que el cálculo de cada volumen es independiente del resto.

2) Problema secuencial: El camino del borracho (The Drunkard walk): Es un problema muy conocido en la literatura de cálculo de probabilidades, y relacionado con los "caminos aleatorios" (random walks [11]): Un borracho camina vacilante por una calle estrecha con riesgo de chocar con sus paredes. Cada paso que da se acerca a alguna de las paredes, y se aleja de la opuesta. Si inicialmente se haya a cinco pasos de distancia de cada pared, se trata de calcular la probabilidad de que choque con alguna pared después de realizar un número de pasos determinados al azar. Dado que los estudiantes aún no conocían las funciones aleatorias, se les suministró una lista con los pasos aleatorios ya calculados, y se les pidió construir una función que nos diga si el borracho choca o no con alguna pared.

La lista de pasos tiene la siguiente forma:

```
(1 0 0 1 0 1 1 1 0 0 1)
```

```
1:  Dar un paso a la
derecha.

0:  Dar un paso a la
izquierda.
```

Mostramos a continuación una de las soluciones programadas por una de estudiante de 13 años para el cálculo de volúmenes:

```
(defun volume (r-list h-list)
  (mapcar 'area r-list h-list))

(defun area (r)
  (* 3.14159 r r))
```

Aunque podría igualmente construirse una función recursiva, la estudiante eligió adecuadamente utilizar map, dada la naturaleza intrínsecamente paralela del problema, con datos independientes. Aunque la estudiante no es consciente aún de la importancia del cómputo paralelo, el código escrito está listo para ser ejecutado en secuencial y en paralelo con mínimos cambios. De hecho, basta importar una librería de programación paralela y utilizar *pmapcar* en lugar de *mapcar* para poder utilizar varios núcleos del computador durante el cálculo.

Después de analizar el segundo problema, la estudiante se dio cuenta de la necesaria ejecución en secuencia de los pasos a repetir, por lo que construyó la siguiente solución recursiva:

```
(defun drunkards-walk (steps-list)
  (check-the-random-walk
    (steps-list 0))) ;begin at
                     ;position '0

(defun check-the-random-walk
                  (steps-list n)
  (if (null steps-list)
      'congratulations
      (if (or (= n 5)(= n -5))
          'bad-luck
        (if (= 0 (car steps-list))
            (check-the-random-walk
              (cdr steps-list) (-1 n))
            (check-the-random-walk
                 (cdr steps-list)
                 (+1 n))))))
```

Lo primero que observamos en las soluciones anteriores es que cuando las dos opciones son válidas para resolver un problema, el estudiante elige la opción basada en *map*. La razón es que es mucho más simple que la solución recursiva; y sólo cuando esta opción no sirve para resolver el problema, recurre a la versión recursiva. Y este era realmente el objetivo: las funciones map son conceptualmente paralelas, por lo que los estudiantes están desarrollando una lógica abstracta de programación paralela, y el código que construyen, que funciona en modo secuencial, es también paralelo, y podrá ejecutarse en entornos paralelos con cambios.

Por otro lado, el modelo ha conseguido que los estudiantes entiendan y apliquen desde el principio la lógica de la programación recursiva. Lo que

tradicionalmente se ha visto como uno de los conceptos más difíciles, es aplicado con normalidad por estudiantes de secundaria después de asistir a un curso de 40 horas de programación.

Aunque estos estudiantes no han estudiado aún herramientas o librerías de programación paralelas, ni conocen las características de los sistemas multi-core, el modo en que han aprendido a programar les permitirá más fácilmente desarrollar otras técnicas que encajan de forma natural con el modelo de programación funcional que conocen.

VI. CONCLUSIONES.

Este artículo propone la programación funcional como la metodología adecuada para enseñar programación a estudiantes que aún no han tomado contacto con la misma. Además, propone la supresión de los *bucles* como herramienta de programación para realizar procesos repetitivos. El motivo es que el bucle impide entender adecuadamente las dependencias que puedan existir entre los datos que se procesan, y que dificulta en el futuro aplicar modelos de programación paralelos a los estudiantes. Las evidencias tomadas de competiciones de programación, asignaturas obligatorias de titulaciones afines y trabajos fin de grado de la Universidad de Extremadura, corroboran lo que ya ha sido destacado en estudios previos: la programación paralela y la recursividad son rehuidas sistemáticamente.

Por otro lado, cuando los bucles se suprimen en la programación funcional, y disponemos tanto de los procesos recursivos como de funciones de orden superior, los estudiantes necesariamente aprenderán dos formas de resolver las tareas repetitivas: (i) basadas en *map* cuando los datos son independientes, y por tanto por naturaleza pueden procesarse con modelo SIMD; (ii) de manera recursiva cuando las tareas tienen dependencias y deben ejecutarse en un cierto orden. El código escrito, que es paralelo per se, y que incluirá en cada caso lo que sea necesario, podrá ser ejecutado en paralelo con sólo incluir la librería de programación paralela necesaria.

Para probar la metodología se ha trabajado con un grupo de estudiantes de secundaria, y los resultados obtenidos muestran el interés de la propuesta.

La prohibición de bucles puede ser aplicada en cualquier lenguaje de programación que incluya funciones de alto orden (high order functions), aunque es recomendable el paradigma funcional, dada las características que le distinguen de sus alternativas.

AGRADECIMIENTOS

Queremos agradecer al Ministerio de Economía y Competitividad, proyecto TIN2017-85727-C4-4-P Deep-Bio- Uex; Junta de Extremadura, Departamento de Comercio y Economía, proyecto G15068 IB16035 cofinanciado por Fondos FEDER: ”Una manera de construir Europa”.

REFERENCIAS

[1] Lahtinen, E., Ala-Mutka, K., & Jrvinen, H. M. (2005, June). A study of the difficulties of novice programmers. InAcm Sigcse Bulletin(Vol. 37, No. 3, pp. 14-18). ACM.

[2] Bruce, K. B. (2005). Controversy on how to teach CS 1: a discussion on the SIGCSE-members mailing list.ACM SIGCSE Bulletin,37(2), 111- 117.

[3] Turbak, F., Royden, C., Stephan, J., & Herbst, J. (1999). Teaching recur- sion before loops in CS1.Journal of Computing in Small Colleges,14(4), 86-101.

[4] Wadler, P. (1998). Why no one uses functional languages. ACM Sigplan Notices, 33(8), 23-27.

[5] Lahtinen, E., Ala-Mutka, K., & Jrvinen, H. M. (2005, June). A study of the difficulties of novice programmers. In Acm Sigcse Bulletin (Vol. 37, No. 3, pp. 14-18). ACM.

[6] Church, A. (1941). The calculi of lambda-conversion (No. 6). Princeton University Press.

[7] Kahney, H. (1983, December). What do novice programmers know about recursion. In Proceedings of the SIGCHI conference on Human Factors in Computing Systems (pp. 235-239). ACM.

[8] McKenney, P. E., Michael, M. M., Gupta, M., Howard, P. W., Triplett, J., & Walpole, J. (2009). Is parallel programming hard, and if so, why?.

[9] KALELIOGLU, F., & Glbahar, Y. (2014). The Effects of Teaching Programming via Scratch on Problem Solving Skills: A Discussion from Learners' Perspective. Informatics in Education, 13(1).

[10] Fernández de Vega, F., Chávez, F., García, M.C. (2018). On the impact of STEM sustained actions on the future of young students. IEEE FIE 2018.

[11] Weiss, G. H. (1983). Random Walks and Their Applications: Widely used as mathematical models, random walks play an important role in several areas of physics, chemistry, and biology. American Scientist, 71(1), 65-71.

[12] Drasch, F. J. (1987). The clisp Programming Environment. Drasch Computer Software.

[13] Fernández de Vega, F. (2019) To be or not To be: That is the Recursive Question. IEEE EDUCON 2019. Dubai, April 2019.

Generación automática de trabajos en grupo para asignaturas de Fundamentos de Computadores y Redes

Joaquín Entrialgo[1], Rubén Usamentiaga[2], Julio Molleda[3], María Teresa González [4] y Daniel F. García [5]

Resumen— **En este artículo se presenta una herramienta para la generación automática de trabajos en grupo para las asignaturas de Fundamentos de Computadores y Redes de la Universidad de Oviedo. Estas asignaturas, que se imparten en los grados de Ingeniería Informática de Gijón y Oviedo, tienen como parte de su metodología el desarrollo de trabajos en grupo. Resulta muy complejo tener trabajos que ayuden a los estudiantes a alcanzar los resultados de aprendizaje y sean, al mismo tiempo, interesantes para ellos, distintos para cada grupo y objetivamente evaluables. Este último punto tiene una especial importancia ya que la asignatura es impartida por un elevado número de docentes, de dos áreas de conocimiento distintas, y es deseable que haya criterios comunes entre todos. La experiencia de cursos anteriores ha mostrado que lograr estos objetivos requiere un gran esfuerzo de preparación y coordinación por parte de todos los profesores implicados. Para tratar de superar estas dificultades, en este proyecto se ha creado una herramienta on-line que, por un lado, genera problemas enfocados a los Fundamentos de Computadores y Redes distintos para cada grupo de estudiantes y, por otro, facilita a los profesores las tareas de preparación y coordinación. La herramienta consiste en una aplicación web que, a partir de unas plantillas que pueden ser actualizadas a principio de cada curso, genera problemas distintos para cada grupo de estudiantes variando algunos elementos indicados en las plantillas. La herramienta ayuda, además, en la corrección, proporcionando respuestas adecuadas a cada problema particular. Esto facilita a los profesores la evaluación y redunda en una mayor homogeneidad de criterios.**

Palabras clave— **Fundamentos de Computadores, Trabajos en grupo, Generador de problemas, Docencia.**

I. Introducción

PARTE de la metodología de las asignaturas de Fundamentos de Computadores y Redes (FCR) que se imparten en el primer curso de los Grados en Ingeniería Informática de la Universidad de Oviedo consiste en utilizar trabajos en grupo. El enfoque pedagógico general seguido en estas asignaturas es constructivista [1], un enfoque ampliamente utilizado en la actualidad y, en particular, en la docencia de materias científicas [2]. Siguiendo la memoria de verificación de los correspondientes grados, estas asignaturas incorporan como parte de su metodología el trabajo en grupo, que permite desarrollar competencias de búsqueda, selección, organización y valoración de la información, adaptación y aplicación a situaciones reales, interpersonales y de organización y gestión [3]. A la hora de organizar esta forma de aprendizaje colaborativo, se han de tener en cuenta tres etapas: etapa de diseño del trabajo, etapa de realización del trabajo y etapa de resultados y realimentación [4]. La herramienta desarrollada en este proyecto facilita estas tres etapas.

Las asignaturas de Fundamentos de Computadores y Redes proporcionan unos conocimientos que, como su propio nombre indica, son fundamentales. Esto hace que, en ocasiones, a los estudiantes les resulte complicado ver su aplicación práctica. Típicamente, esta materia se explica diseñando un computador a partir de principios básicos; sin embargo, la mayor parte de los estudiantes no desarrollarán su carrera profesional diseñando ordenadores, sino utilizándolos a través de la programación en lenguajes de alto nivel. Un enfoque propuesto por dos profesores de la Carnegie Mellon University [5] consiste en cambiar la perspectiva y centrar el estudio de los computadores desde el punto de vista del programador. Una de las actividades que proponen siguiendo este enfoque es que los estudiantes utilicen los conocimientos adquiridos en la asignatura para desactivar "bombas binarias": pequeños programas que piden unas entradas y, si no se introducen las adecuadas, muestran un mensaje diciendo que han estallado. Los estudiantes no disponen del código en alto nivel, sólo del código máquina, por lo que tienen que utilizar los conocimientos de bajo nivel adquiridos en la asignatura para determinar qué entradas evitan que las bombas estallen. Este proceso de ingeniería inversa es además habitual en la práctica profesional de la seguridad informática, dentro del análisis forense, lo que proporciona a los estudiantes una visión práctica y aplicada de los conocimientos.

En el curso 2016-2017 se introdujo un trabajo en grupo de este tipo en las asignaturas de FCR de la Universidad de Oviedo. El trabajo permitió que los estudiantes alcanzasen los resultados de aprendizaje esperados. Sin embargo, supuso un gran esfuerzo para los profesores y presentó algunos problemas de comprensión del enunciado para los estudiantes

[1]Dpto. de Informática, Universidad de Oviedo, e-mail: joaquin@uniovi.es.
[2]Dpto. de Informática, Universidad de Oviedo, e-mail: rusamentiaga@uniovi.es.
[3]Dpto. de Informática, Universidad de Oviedo, e-mail: jmolleda@uniovi.es.
[4]Dpto. de Informática, Universidad de Oviedo, e-mail: maytega@uniovi.es.
[5]Dpto. de Informática, Universidad de Oviedo, e-mail: dfgarcia@uniovi.es.

porque estaba diseñado de manera parametrizada, de tal forma que cada grupo tenía un problema distinto, lo que complicaba la explicación. Por otra parte, los profesores tenían que generar bombas binarias distintas para cada grupo, un proceso tedioso, realizado de forma manual, que consume mucho tiempo y es propenso a errores. Además, la distribución de los ficheros necesarios para cada grupo no estaba organizada y cada profesor utilizó el método que consideró adecuado.

Por esta razón se realizó un Proyecto de Innovación Docente para desarrollar una herramienta que permitiera utilizar aprendizaje mejorado con tecnología (Technology-Enhanced Learning o TEL [6]), de tal manera que se facilitara el proceso tanto para el profesor como para el estudiante, permitiendo además una mejor monitorización del desarrollo del trabajo [7].

II. Metodología

El primer paso seguido fue definir problemas parametrizables para el trabajo en grupo de las asignaturas de FCR. Experiencias de cursos anteriores demostraron que era posible un modelo basado en las bombas binarias propuestas en [5], pero siendo necesario adaptarlas al contexto de las asignaturas FCR y los objetivos propuestos para el trabajo en grupo, que son los siguientes:

- Aplicar los conocimientos de la asignatura en un entorno realista, aunque simplificado. Los estudiantes comprobarán cómo los conocimientos de la asignatura son fundamentales para la seguridad informática.
- Entrar en contacto con una arquitectura real: la arquitectura Intel x86-32. En la parte teórica y de prácticas de laboratorio de la asignatura se utiliza una arquitectura que no está implementada en la realidad y se denomina CT (Computador Teórico) [8]. Al conocer una segunda arquitectura, los estudiantes podrán comprobar cómo conceptos generales se pueden aplicar de distintas maneras en implementaciones concretas.
- Practicar el lenguaje C/C++, necesario para la parte teórica y de prácticas de laboratorio de la asignatura y para asignaturas futuras, además de ser un lenguaje muy usado en la práctica. Asimismo, el estudiante verá cómo mezclarlo con ensamblador.
- Practicar el uso de máscaras y desplazamientos, elementos muy habituales en el software de sistemas, seguridad, tratamiento de imágenes o comunicaciones.
- Desarrollar habilidades de trabajo en grupo y documentación básicas para un Ingeniero en Informática.

A partir de estos objetivos y teniendo en cuenta cómo se introducen los distintos conocimientos fundamentales en la planificación de la asignatura, se decide dividir el trabajo en dos fases, que se detallan a continuación.

A. *Fase 1. Realización de un programa en C/C++ y análisis de su funcionamiento*

Esta fase se plantea a modo de introducción para poder llevar a cabo el análisis de las bombas binarias en la siguiente fase.

Los estudiantes tendrán que realizar un programa en C/C++ con varias funciones cuya especificación será distinta para cada grupo. Tres funciones deben llevar a cabo operaciones con máscaras de bits. Los bits involucrados serán el parámetro que cambiará para cada grupo. Cada función usa un enfoque distinto: la primera utiliza directamente C, la segunda utiliza ensamblador incrustado en C y la tercera llama desde C a un procedimiento definido en un archivo de ensamblador. También habrá que realizar una función que determine si una cadena introducida es igual a otra cadena que será parametrizable para cada grupo.

Los estudiantes deben además llevar a cabo tareas de análisis del programa desarrollado, tales como encontrar entradas válidas e inválidas y explicar cómo se han traducido ciertos elementos de C/C++ a código máquina.

En la Figura 1 se resume el funcionamiento de la fase 1, que comienza creando un curso para cada escuela en la herramienta y creando los usuarios. Estas tareas son desarrolladas por el administrador de la herramienta, que será uno de los profesores. Para facilitar la creación de usuarios, la herramienta permitirá importar listas de alumnos en formato CSV, de tal manera que automáticamente se crearán los usuarios correspondientes a los estudiantes y se asignarán a grupos de Prácticas de Laboratorio (PL). El administrador tendrá que crear manualmente los usuarios para los profesores y asignarlos a los PL correspondientes. A continuación, la herramienta permitirá a los profesores formar los grupos de trabajo (ver Figura 2) y gestionarlos (ver Figura 3). Los estudiantes podrán obtener las instrucciones parametrizadas para cada grupo (ver Figura 4) y descargar los ficheros de proyecto básicos para empezar a programar. Los estudiantes entregarán el proyecto de programación realizado y una memoria. La entrega se hará a través del Campus Virtual para centralizar las entregas de todas las tareas de la asignatura en la misma plataforma. Finalmente, durante la etapa de evaluación, los profesores podrán obtener a través de la herramienta desarrollada entradas válidas e inválidas para probar los programas desarrollados por los estudiantes, así como comprobar si las entradas válidas e inválidas proporcionadas por los estudiantes son correctas (ver Figura 5).

B. *Fase 2. Análisis de bombas binarias*

En esta fase se plantea que se dispone de una serie de bombas binarias desarrolladas por distintas organizaciones criminales. Cada grupo puede descargar desde la herramienta una bomba

Fig. 1. Esquema de la fase 1.

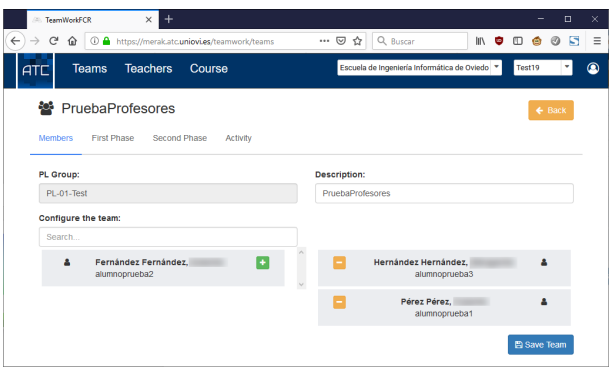


Fig. 2. Pantalla de formación de grupos para el profesor.

binaria y debe encontrar la forma de desactivarla. La bomba binaria consiste en un programa ejecutable en sistemas operativos Windows. Este programa tiene tres etapas. En cada una pide una entrada. Si cumple ciertos criterios, la etapa se desactiva y se pasa a la siguiente; si no, la bomba estalla. Si se consiguen desactivar todas las etapas, la bomba será desactivada.

Las bombas binarias se conectan a un servidor del que reciben mensajes, de manera similar a como el *malware* real se conecta a los servidores de mando y control (*Command and Control*, en inglés, usualmente abreviado C&C). En este caso será un mensaje muy sencillo que incluirá una cadena que permite identificar a la organización criminal que desarrolló la bomba. Los estudiantes deben encontrar este nombre utilizando los conocimientos y las herramientas relacionadas con fundamentos de redes de computadores que se explican en la asignatura.

La Figura 6 resume el esquema de esta Fase 2, que comienza con los estudiantes descargando las bombas binarias generadas automáticamente por la herramienta para cada grupo de trabajo. A

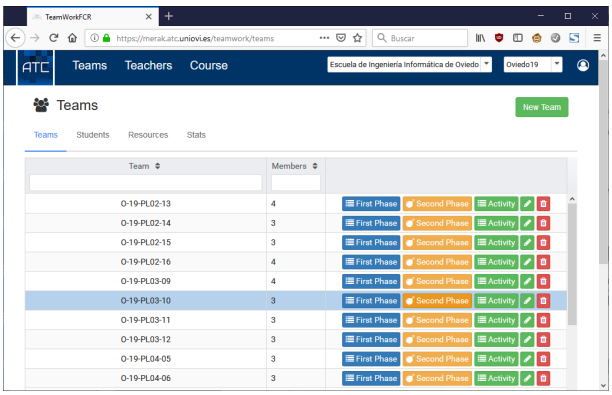


Fig. 3. Lista de grupos. Vista para el profesor.

continuación, los estudiantes realizarán el análisis de las bombas buscando las entradas para desactivarlas. Deben entregar una memoria y una bomba modificada para que no estalle sin requerir ninguna entrada. Esta entrega será nuevamente en el Campus Virtual para centralizar todas las entregas de los alumnos en una plataforma. Finalmente, el profesor evaluará el desempeño de los estudiantes en esta fase analizando los ficheros subidos y los datos para cada grupo que indica la herramienta.

Los parámetros distintos para cada grupo serán las operaciones que se hacen en cada etapa y el nombre de la organización criminal. La herramienta permite obtener de manera automática una bomba distinta para cada grupo (ver Figura 7). El profesor además podrá descargar tanto los ficheros que se distribuyen a cada grupo de alumnos como una versión que incluye el código fuente, lo que le facilitará la evaluación del trabajo.

Cuando una fase es desactivada, se manda un mensaje al servidor de la herramienta. De esta manera tanto los alumnos como los profesores podrán monitorizar cómo avanza esta segunda fase a lo largo del tiempo, como también se muestra en la Figura 7.

Se decidió implementar la herramienta como una aplicación web utilizando la tecnología ASP.NET de Microsoft, ya que los archivos ejecutables generados deberían ser compatibles con Windows y Visual Studio, tecnologías que se utilizan en la asignatura para el desarrollo de programas y su depuración.

III. Resultados

Para valorar la experiencia, se han utilizado diversos indicadores: encuestas tanto a profesores como a estudiantes, registros de uso de la aplicación y notas obtenidas. A continuación, se describen cada uno de ellos.

A. Encuestas

Se llevó a cabo una encuesta a profesores con diversas preguntas sobre la experiencia. Los 11 profesores de la asignatura respondieron a la encuesta. Se llevó a cabo también una encuesta para los estudiantes que fue contestada por 28 estudiantes en Gijón y 43 en Oviedo. Los resultados se resumen en la Figura 8.

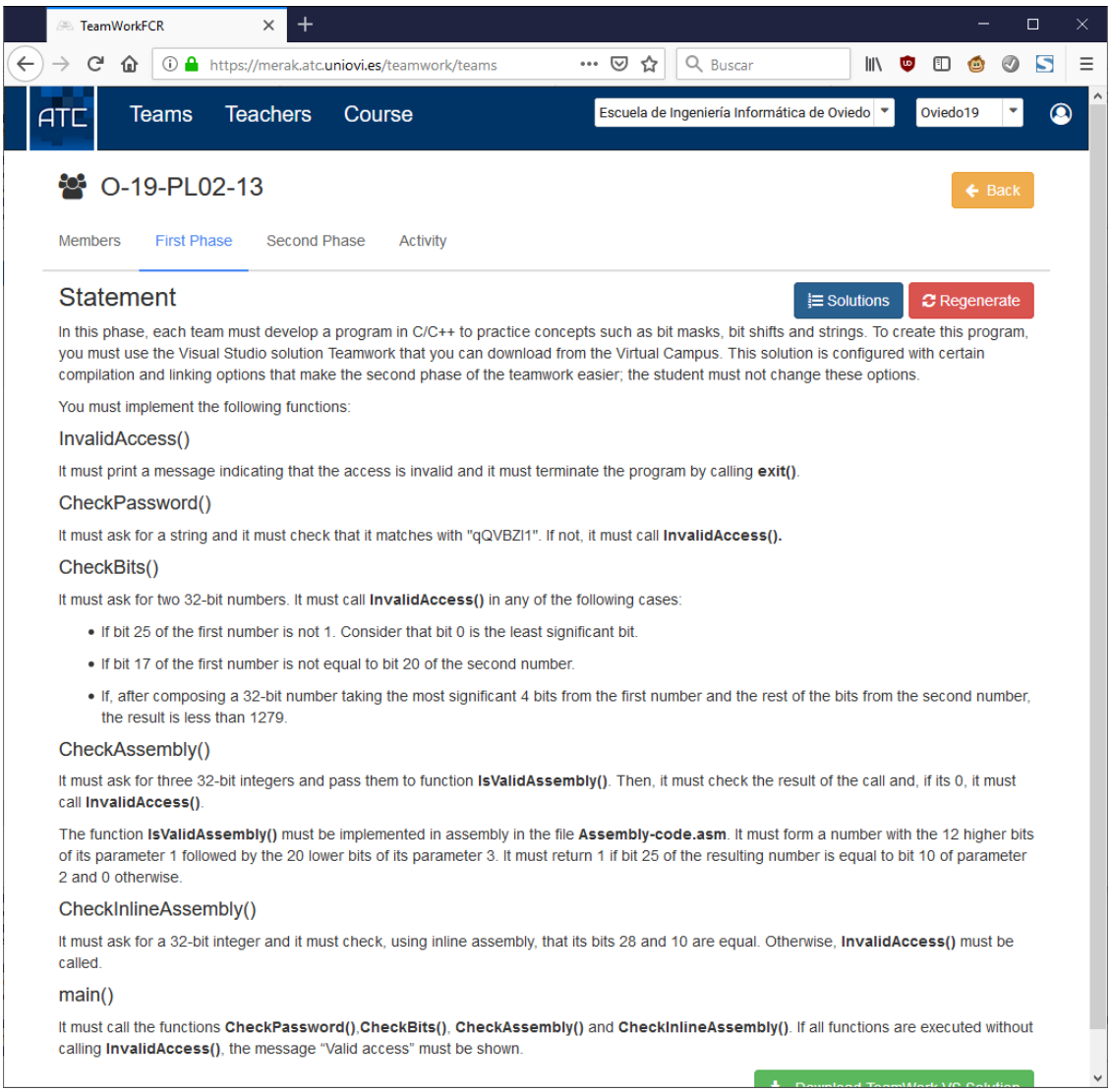


Fig. 4. Instrucciones parametrizadas para la fase 1 de un grupo.

La primera pregunta valoraba la usabilidad de la herramienta desarrollada. Los resultados obtenidos fueron superiores a 8, en una escala de 0 a 10 puntos, tanto para los profesores como los estudiantes.

La segunda pregunta permitía valorar cuánto aportaba la realización del trabajo en grupo al aprendizaje. Tanto profesores como alumnos lo valoraron con un notable. Los 10 profesores que habían impartido la asignatura el curso anterior valoraron la mejora con respecto al año pasado con un 9.8. Entre los estudiantes, 8 de Gijón y otros 8 de Oviedo de los que contestaron la encuesta la habían cursado el año anterior; la media de mejora fue valorada con un 7.6.

También se realizaron dos preguntas para valorar la dificultad de realización de las fases del trabajo. Como se puede observar, tanto profesores como alumnos consideran más compleja la fase 2.

Además de las preguntas comunes a profesores y alumnos reflejadas en la Figura 8, se hicieron dos preguntas sólo a los profesores para valorar la dificultad en gestionar las fases, obteniendo un valor de 3.8 para la fase 1 y 4.7 para la fase 2, lo que indica un grado de dificultad de gestión bajo.

Por último, se realizaron también preguntas de formato abierto, cuyos resultados se comentan en la sección de Conclusiones.

B. Registros de uso

Se llevó a cabo un análisis de los registros de uso de la aplicación. En base a este análisis se ha podido determinar que hay una media de 16.76 accesos por usuario. La Figura 9 muestra cómo se distribuyeron estos accesos a lo largo del tiempo, observándose un incremento del uso cuando se acercaban las fechas de entrega en alguna de las dos fases.

C. Notas de los alumnos en la parte de trabajo en grupo

La Figura 10 muestra las notas obtenidas en el trabajo en grupo y en la asignatura comparando 2017 y 2018. La nota media en el trabajo en grupo pasó de 5.2 a 5.6, lo que se considera aceptable. Se puede observar que es mejor que la nota obtenida de media en la asignatura y que en Gijón no se ha notado un cambio significativo, mientras que en Oviedo hay

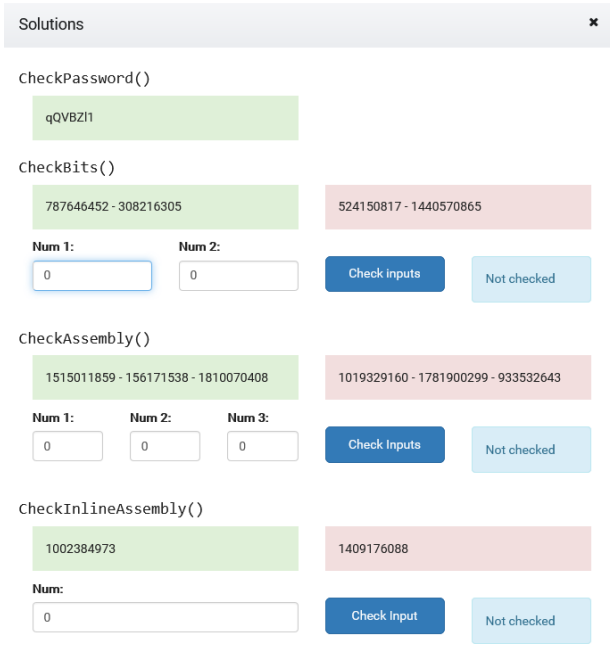


Fig. 5. Soluciones parametrizadas para la fase 1 de un grupo.

Fig. 6. Esquema de la Fase 2.

casi dos puntos de diferencia. Hay que tener en cuenta que en la escuela de Gijón pudieron influir otros factores, como un cambio casi completo en los profesores que la impartieron.

IV. Conclusiones

En este artículo se ha presentado una herramienta para generar automáticamente trabajos en grupo en las asignaturas de Fundamentos de Computadores y Redes y su implantación durante el curso 2017-2018.

El trabajo desarrollado se basa en el concepto de bombas binarias presentado en [5]. Se ha extendido este trabajo para integrarlo con los objetivos de las citadas asignaturas. La herramienta desarrollada ha facilitado la coordinación entre el numeroso grupo de profesores que imparte las asignaturas. Además, para los alumnos ha supuesto tener un repositorio centralizado de fácil acceso con la información para el trabajo en grupo.

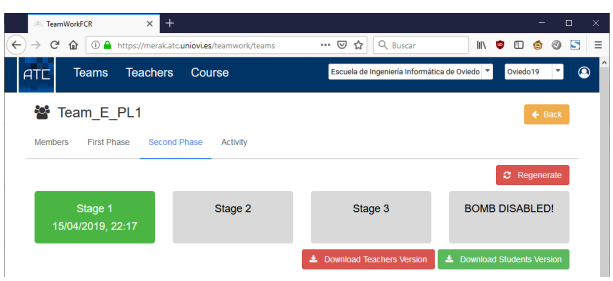


Fig. 7. Pantalla de obtención de bombas y seguimiento para la fase 2.

Los resultados de los indicadores muestran que la experiencia ha sido exitosa: tanto los alumnos como los profesores han valorado con una puntuación muy alta la herramienta desarrollada. Asimismo, los registros de acceso muestran una amplia utilización de la herramienta y las notas obtenidas por los alumnos demuestran que el resultado del trabajo en grupo ha sido positivo.

En las preguntas abiertas de las encuestas, los profesores destacan como puntos fuertes que se ha facilitado la gestión del trabajo en grupo y que éste ha servido para dar una visión más práctica de la asignatura a los alumnos y les ha ayudado a comprender los conceptos que en ella se estudian.

Como puntos débiles, los profesores han encontrado que uno de los problemas generados en la fase 1 era excesivamente similar para todos los grupos. Esto se podría solucionar cambiando las plantillas para esta parte, de forma que el problema generado para cada grupo se diferenciara más. También se señaló que se podrían incrementar las ayudas a la corrección para el profesor, aportando mayor automatización. Esto se podría llevar a cabo, pero habría que tener en cuenta los riesgos de seguridad que implicaría para el servidor donde se ejecuta la herramienta el ejecutar código de los alumnos.

Tanto estudiantes como profesores han encontrado también en ocasiones problemas de organización de los grupos de trabajo, en general relacionados con los estudiantes que abandonan la asignatura y, en consecuencia, su grupo de trabajo. Algunos estudiantes han indicado también que desearían más tiempo para la segunda fase y más ayuda en clase. Estos problemas no son achacables a la herramienta desarrollada.

Durante el curso 2019-2020 se está utilizando esta herramienta, incorporando algunas mejoras a partir de los comentarios del curso anterior. En concreto, se ha modificado el problema de la fase 1 en el que se habían encontrado demasiadas similitudes entre grupos. Por otra parte, se ha añadido una pantalla para mostrar las soluciones de la fase 2 y, de esta forma, facilitar su corrección.

Agradecimientos

La herramienta ha sido desarrollada por Rubén Urbano Colmenar en su Trabajo Fin de Grado y dentro del Proyecto de Innovación Docente de la Universidad de Oviedo “Generación automática de trabajos en grupo para Ingeniería Informática en el ámbito de Computadores y Redes (PINN-17-A-046)”.

Referencias

[1] Keith S Taber, “Constructivism as educational theory: Contingency in learning, and optimally guided instruction,” *Educational Theory*, 2012.

[2] Mustafa Cakir, “Constructivist approaches to learning in science and their implications for science pedagogy: A literature review.,” *International journal of environmental and science education*, vol. 3, no. 4, pp. 193–206, 2008.

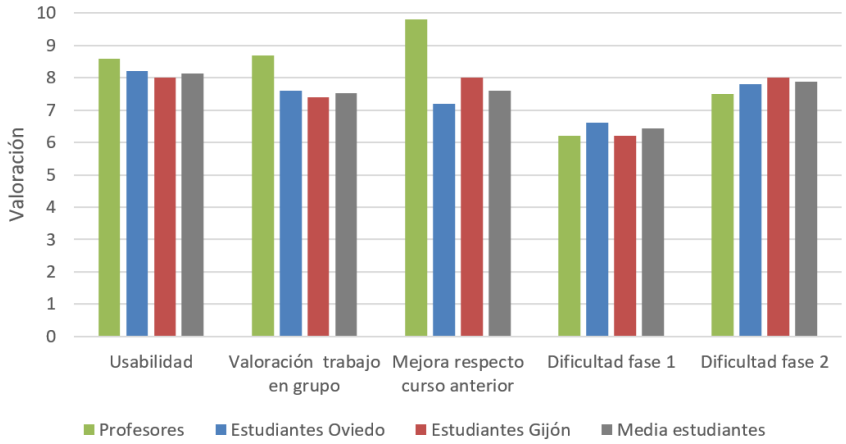


Fig. 8. Resumen encuestas.

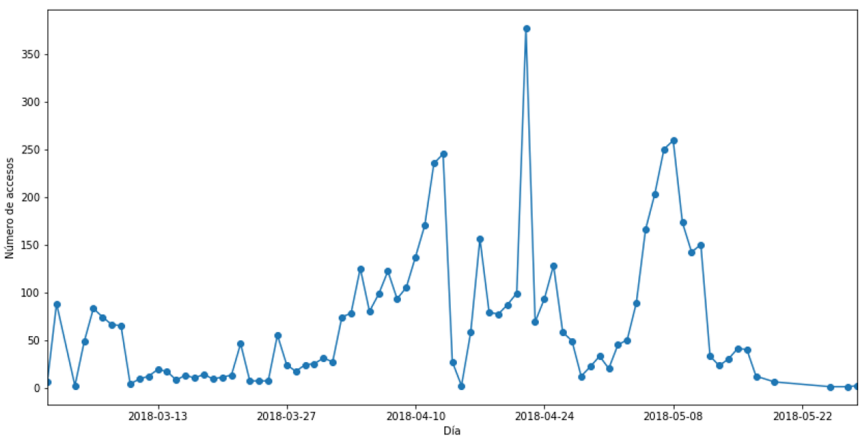


Fig. 9. Número de accesos por día.

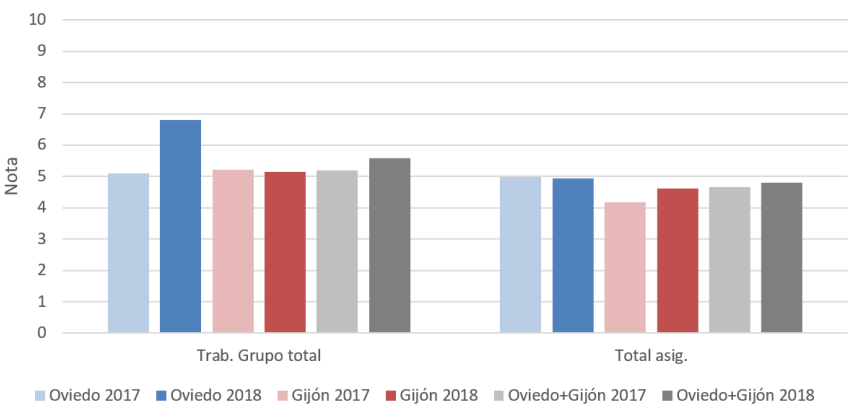


Fig. 10. Notas obtenidas en el trabajo en grupo y en la asignatura.

[3] David Johnson, Roger T. Johnson, and Karl Smith, "Active learning: Cooperation in the college classroom," 01 1998.

[4] Junko Shimazoe and Howard Aldrich, "Group work can be gratifying: Understanding & overcoming resistance to cooperative learning," *College teaching*, vol. 58, no. 2, pp. 52–57, 2010.

[5] Randal E Bryant and David R O'Hallaron, "Introducing computer systems from a programmer's perspective," in *ACM SIGCSE Bulletin*. ACM, 2001, vol. 33, pp. 90–94.

[6] Antoni Badia, "Research trends in technology-enhanced learning," *Infancia y Aprendizaje*, vol. 38, no. 2, pp. 253–278, 2015.

[7] Maria Jesus Rodriguez Triana, Luis Pablo Prieto Santos, Andrii Vozniuk, Mina Shirvani Boroujeni, Beat Adrian Schwendimann, Adrian Christian Holzer, and Denis Gillet, "Monitoring, awareness and reflection in blended technology enhanced learning: a systematic review," *International Journal of Technology Enhanced Learning*, vol. 9, pp. 126–150, 2017.

[8] J. Entrialgo, J.L. Díaz, A.M. Campos, and D.F. García Martínez, "Simulador educacional de un computador elemental basado en la arquitectura von neumann," in *XIII Jornadas de Paralelismo: Lleida, 9, 10 y 11 de septiembre de 2002: actas*. Universitat de Lleida, 2002, pp. 95–98.

Herramienta software para la enseñanza del paralelismo a nivel de instrucciones (ILP)

Manuel Rivas-Pérez*, Manuel Domínguez-Morales*, Alejandro Linares-Barranco*, Antón Civit-Balcells*

Resumen—**La arquitectura de computadores es una asignatura de gran importancia en las titulaciones de Informática. Debido a la dificultad de poder acceder directamente a los componentes internos de un procesador, es común en la enseñanza hacer uso de una o varias herramientas software que simulen el comportamiento interno del mismo. Sin embargo, suelen ser herramientas limitadas u obsoletas. Esto provoca una gran dificultad a la hora de transmitir los conocimientos en este tipo de asignaturas. Este caso se puede aplicar directamente al paralelismo a nivel de instrucciones (ILP), que precisa poder visualizar el comportamiento a nivel interno del camino de datos del procesador. Para ello, en este trabajo se presenta una herramienta software desarrollada íntegramente a medida para la enseñanza de arquitectura de computadores, que permite la visualización detallada del pipeline del procesador, permitiendo modificaciones sobre su ejecución y aplicar optimizaciones en tiempo de ejecución para poder obtener medidas de rendimiento. Seguidamente, tras tres años de uso en docencia, se presenta un estudio sobre la bondad del uso de la herramienta atendiendo a la evolución de las calificaciones y a encuestas realizadas sobre el alumnado.**

Palabras clave—**ILP, segmentación, arquitectura de computadores, enseñanza, MIPS32.**

I. INTRODUCCIÓN

La electrónica digital ha evolucionado rápidamente en muy poco tiempo llegando a integrar más de mil millones de transistores en un mismo chip. La arquitectura también ha evolucionado con nuevas estructuras como la segmentación, el uso de cachés, procesamiento multihilo o el multiprocesamiento ayudados por este aumento de la integración de los transistores. El estudio de la arquitectura de los computadores es de gran importancia en las titulaciones universitarias relacionadas con la informática. Estas arquitecturas, fruto de su evolución, pueden resultar complejas lo que provoca que muchos estudiantes encuentren dificultades para comprender algunos aspectos de la asignatura de arquitectura de computadores, como la segmentación de cauce (*pipelining*) o el cálculo del rendimiento del procesador. La segmentación es una técnica muy importante en arquitectura de computadores en la que varias instrucciones pueden ejecutarse simultáneamente manteniendo cada una de ellas en una etapa diferente de la ruta de datos (*datapath*). El uso de procesadores ejemplo ayuda al alumno a comprender los conceptos clave de la arquitectura y las mejoras de los procesadores. Este es el caso del procesador MIPS descrito en los libros de Henessy y Patterson [1], [2] ampliamente conocidos por la comunidad universitaria, que han convertido a este procesador de ejemplo en uno de los más utilizados para enseñar la asignatura de arquitectura de computadores en las universidades [3].

**Dpto. Arquitectura y Tecnología de Computadores, E.T.S. Ingeniería Informática, Universidad de Sevilla.*

Pero, aun así, estos contenidos son difíciles de asimilar si se exponen únicamente haciendo uso de la pizarra o con lápiz y papel. Es por ello por lo que muchos los profesores emplean simuladores gráficos como herramienta para mostrar los conceptos de forma intuitiva e interactiva simplificando la forma en la que los profesores enseñan y los alumnos aprenden la arquitectura de computadores. Hay numerosos artículos que tratan sobre el aprendizaje de la arquitectura de computadores [4]–[7] y la mayoría defienden la importancia de los simuladores para el aprendizaje. De hecho, estos simuladores se han convertido en una herramienta indispensable para que los alumnos puedan entender las complejas arquitecturas difíciles de asimilar sin la interfaz gráfica que los simuladores actuales pueden ofrecer [8].

En este trabajo se presenta una herramienta software que simula y visualiza el procesamiento paralelo de instrucciones del procesador segmentado MIPS para mejorar la calidad de la enseñanza y dar a los estudiantes un entorno en el que experimentar. Este trabajo se organiza de la siguiente forma: En la sección 2 se describe la implementación y características del simulador; en la sección 3, se expone el funcionamiento del entorno visual VisualMIPS32; a continuación, se evalúa la herramienta en base a las calificaciones del alumnado y a las encuestas de satisfacción; para, finalmente, presentar las mejoras futuras y conclusiones del trabajo.

II. CARACTERÍSTICAS E IMPLEMENTACIÓN

Se describirá en primera instancia las características básicas del procesador MIPS para, posteriormente, introducir los detalles de implementación.

A. Características

Los procesadores MIPS forman parte de una gran familia de procesadores RISC desde su primer procesador R2000 en el 1986 cuyo juego de instrucciones ha evolucionado desde la versión original denominada MIPS I hasta las actuales MIPS32 y MIPS64. Gracias a la sencillez de su arquitectura y al hecho de que actualmente sea bastante utilizado en sistemas embebidos, lo ha convertido en un procesador muy adecuado para el estudio de la arquitectura de computadores en las universidades.

El simulador presentado en este trabajo está basado en el procesador MIPS32 cuya arquitectura ISA puede consultarse en [9]–[11]. Las características principales de este procesador son, entre otras: Procesador RISC segmentado de 32 bits y direcciones de 32 bits, instrucciones de tamaño fijo, 32 registros de propósito general, unidades funcionales independientes para producto y división, detección y control de riesgos, implementa adelantamientos (*forwarding*) y permite

saltos retardaos. Todas estas características han sido implementadas en el procesador siendo configurables la utilización de adelantamientos, los saltos retardados, el número de ciclos que duran las etapas y cuáles de ellas son segmentadas.

De todo el repertorio de instrucciones de la arquitectura MIPS32, formado por algo más de 200 instrucciones incluyendo las que operan en coma flotante, este simulador implementa 147 de ellas que son las que comúnmente se encuentran en la mayoría de los programas en ensamblador:

- Todas las cargas con y sin signo del MIPS I.
- Todos los almacenamientos con y sin signo del MIPS I.
- Todas las instrucciones aritméticas y lógicas con enteros del MIPS I.
- Gran parte de las instrucciones aritméticas y lógicas en coma flotante del MIPS I.
- Gran parte de las instrucciones de control del MIPS I.

B. Implementación

Esta herramienta se ha implementado en C# .NET y consta fundamentalmente de los siguientes elementos:

- Ensamblador: Se ha desarrollado una librería dinámica (.dll) encargada de analizar el código ensamblador y generar el código máquina incluyendo la información del analizador.
- Simulador MIPS32: Se ha implementado el simulador del MIPS32 en otra librería dll independiente incluyendo la memoria principal. Básicamente se encarga de simular paso a paso el procesamiento de las instrucciones e informar de las etapas del pipeline. Esta librería utiliza directamente el componente anterior por lo que abarca todo el proceso desde el ensamblado del programa hasta la simulación.
- ConsolaMips32: Es una interfaz que muestra por pantalla el cronograma de ejecución en modo texto.
- VisualMips32. Interfaz gráfica del simulador que integra el editor y que muestra el estado del procesador desde distintos puntos de vista como es el pipeline, cronograma, registros, memoria, etc. durante la simulación de forma gráfica.

1) Implementación del ensamblador

Los analizadores léxico y sintáctico, así como el generador de código, se encuentran implementados en la librería *AnalizadorMIPS32_ANTLR3.dll* cuyo diagrama de clases se muestra de forma muy simplificada en la Fig. 1. Este simulador verifica la sintaxis de las instrucciones basándose en el repertorio de instrucciones (ISA) descrito en [26] [27]. En la Fig.1 destaca la clase *EnsambladorMIPS* encargada de gestionar el análisis, el control de errores y la generación de código. A partir del programa en ensamblador, la clase *MIPSLexer* es la responsable de realizar el análisis léxico y generar la lista de tokens.

Posteriormente, el analizador sintáctico *MIPSParser* realiza el parsing de los tokens del analizador léxico y genera una lista de declaraciones de instrucciones (clase *DecInsts*). Cada objeto de la clase *DecInst* guarda el token que identifica la operación de la instrucción, una lista de sus parámetros y la dirección que le correspondería en memoria. Durante la fase de generación de código, se crea un objeto de la clase *CodigoFuente* el cual genera una lista de objetos *FormatoInstruccion* a partir de la lista de declaración de instrucciones obtenida por el analizador sintáctico y la clase estática *Kinstruc*. La clase abstracta *FormatoInstruccion* es implementada por cada uno de los formatos de instrucción que posee el MIPS: *FormatoR* (registro), *FormatoI* (inmediato), *FormatoJ* (salto incondicional) y *FormatoFR* (operaciones en punto flotante). *FormatoInstruccion* representa a una instrucción máquina a simular y mantiene, además del código de la instrucción, toda la información relacionada con el programa fuente, los campos de cada formato, dirección de memoria, mnemónico, etc. Aunque podría haberse generado gran parte del código (salvo la resolución de etiquetas) durante el análisis sintáctico, realizar la generación de código en un paso posterior reduce el tiempo del análisis sintáctico y por consiguiente el de la detección de errores del programa.

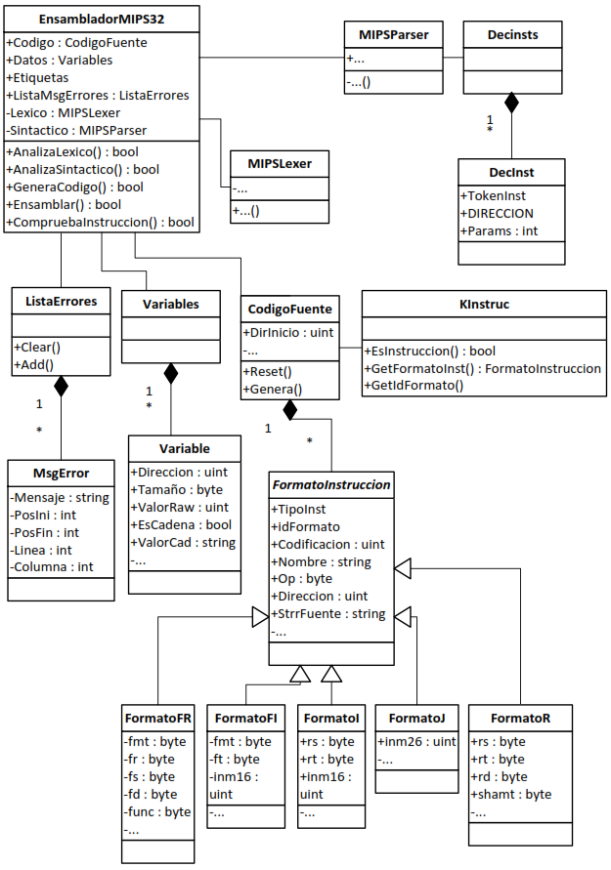


Fig. 1. Diagrama de clases básico del ensamblador.

2) Implementación del simulador

La implementación del simulador está subdividida básicamente en 3 elementos: La memoria, los bancos de registros y el procesador. Este último se encarga de la parte más compleja (implementar cada etapa del pipeline, decodificar instrucciones, implementar los desvíos para los adelantamientos y registrar el estado del pipeline en cada ciclo), permitiendo crear un historial de acontecimientos con el fin de simplificar el desarrollo de

interfaces de usuario. Un diagrama de clases simplificado del simulador se muestra en la Fig. 2. En ella sólo se han representado los atributos, métodos y relaciones entre clases más importantes para mantener la claridad del diagrama.

El sistema simula una memoria de 4GB, esto es, la capacidad máxima admisible para el procesador MIPS32. Puesto que implementar todas las posiciones de la memoria sería muy costoso, se ha optado por diseñar un método que almacene sólo los rangos de posiciones de memoria que contengan valores distintos de cero. Con este método, al escribir datos en memoria fuera de los rangos ya definidos, se crea un nuevo rango de memoria o se amplía un rango ya existente en función de lo alejados que se encuentren los nuevos datos de dicho rango.

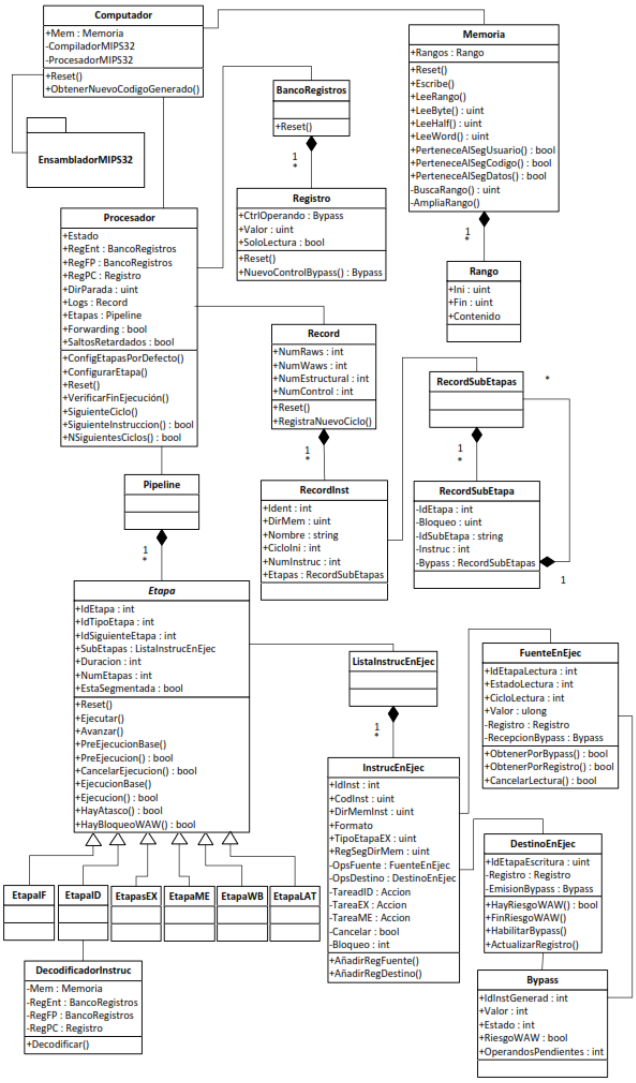


Fig. 2. Diagrama de clases básico del simulador.

La clase Procesador es la responsable de controlar la ejecución paso a paso. Para la simulación de un ciclo del reloj, el procesador realiza un proceso de 3 pasos con cada etapa del pipeline empezando desde la última hasta la primera etapa. Tales pasos se realizan de forma intercalada, esto es, se realiza el paso 1 en todas las etapas antes de continuar con el paso 2. Estos pasos consisten en: Avanzar (transferir la instrucción en ejecución a la siguiente etapa si se encuentra libre), Pre-ejecución (verificar si la instrucción reúne las condiciones para la etapa en la que se encuentra; por ejemplo, disponibilidad de operandos); y Ejecución (verificar condiciones relacionadas con operandos destino y realizar las tareas específicas de esta etapa).

Para la implementación de los desvíos cuando hay adelantamientos (*forwarding*), se utiliza la clase Bypass que es mantenida por Pipeline a través de las clases *Registro*, *FuenteEnEjec* y *DestinoEnEjec*.

La Fig. 3 muestra un ejemplo del funcionamiento del simulador indicándose el instante en el que se crean los desvíos, se suscribe a ellos, se habilitan, se envían operandos por *bypass* y se actualiza el registro. Por ejemplo, el ciclo de vida del *bypass* B1:

- Ciclo 2 (ID de I1): se crea el *bypass* B1 asociado al registro destino $1 de la instrucción 1.
- Ciclo 3 (EX de I1): el *bypass* B1 es habilitado, así pues, el valor de $1 está disponible por bypass.
- Ciclo 3 (ID de I2): el operando fuente $1 de I2 se suscribe a B1 para tomar el valor desde éste cuando la instrucción I2 lo precise.
- Ciclo 4 (EX de I2): la instrucción I2 recibe el dato correspondiente por *bypass* gracias a la suscripción que se realizó a B1.
- Ciclo 4 (ID de I3): el operando destino de la instrucción I3 es $1 por lo que se crea un nuevo *bypass* (bypass B3) que remplaza a B1. A partir de este momento, las nuevas suscripciones a $1 irán dirigidas a B3.
- Ciclo 5 (WB de I1): se actualiza el valor del *bypass* B1 en el registro.

	Ciclo	1	2	3	4	5	6	7
I1	addi $1,$0,1	IF	**ID**	**EX**	ME	WB		
I2	addi $5,$1,3		IF	**ID**	**EX**	ME	WB	
I3	lw $1,0($5)			IF	**ID**	EX	ME	WB
I4	add $9,$1,$7				IF	ID	--	EX
Bypass creado			$1→B1	$5→B2	$1→B3	$9→B4		
Bypass suscrito				B1@I2	B2@I3	B3@I4		
Bypass habilitado				B1	B2		B3	B4
Bypass enviado					B1→I2	B2→I3		B3→I4
Bypass actualizado						B1	B2	B3

Fig. 3. Ejemplo de funcionamiento del simulador.

El simulador implementa un modelo para registrar los sucesos que se van produciendo en cada etapa para cada ciclo de reloj durante la simulación, como son los bloqueos o los desvíos que se activan. El objetivo de mantener este historial de acontecimientos es ofrecer un mayor soporte a las interfaces que podrían necesitar estos datos; por ejemplo, cuando el usuario navegue por el cronograma. Las clases que implementan esta funcionalidad se muestran en la Fig. 2.

La clase *Record* mantiene además de la estadística de rendimiento, una lista de las instrucciones que se han ejecutado. Cada instrucción ejecutada se registra en un objeto de la clase *RecordInst* que incluye toda la información que pudiera ser relevante posteriormente, como es el ciclo en el que comenzó la ejecución, un identificador de la línea con la que se corresponde en el programa ensamblador, la dirección de memoria y una lista de lo ocurrido en cada ciclo de reloj mientras estuvo la instrucción en ejecución. Esta lista de sucesos está implementada en la clase *RecordSubEtapas*, la cual contiene objetos de *RecordSubEtapa* que son los responsables de guardar la información del estado de la instrucción en cada ciclo, como es el tipo de bloqueo

que se produjo o desde o hacia dónde se activó el desvío.

III. VISUALMIPS32

La interfaz gráfica VisualMips32 para Windows ofrece un entorno de simulación integrado con las herramientas necesarias para la edición, depuración y simulación de un código ensamblador para MIPS. El objetivo principal de este entorno de desarrollo es simplificar la interacción con el simulador y visualizar de forma clara e intuitiva el estado del procesador en cualquier momento a través del cronograma de ejecución de las instrucciones, el contenido de la memoria, los registros del procesador y la representación del pipeline.

El entorno se compone fundamentalmente de siete ventanas que el usuario puede acoplar a la ventana principal en forma de pestañas o subdividiendo el área de otras ya acopladas. Al iniciar la aplicación las ventanas se encuentran distribuidas en la ventana principal como muestra la Fig. 4.

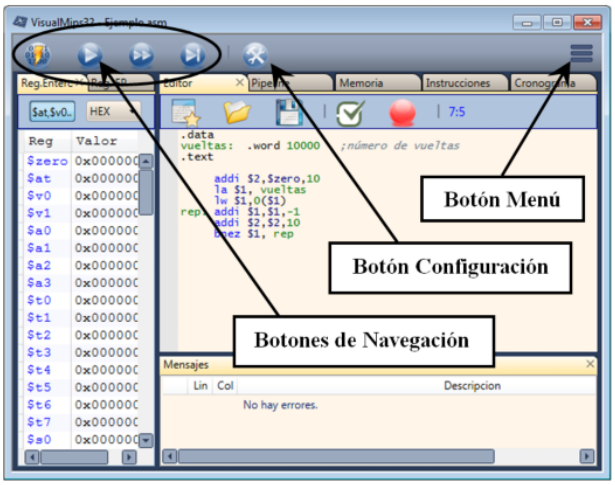


Fig. 4. Visión general del entorno, la ventana principal.

La ventana principal del entorno, que sirve de contenedor para el resto de las ventanas, posee los siguientes botones: Botón de Menú (funciones del entorno), Botón de Configuración (configuración del procesador y simulador), Botones de navegación (reiniciar y avanzar en la simulación).

A. *Ventana Editor*

Esta venta se utiliza principalmente para cargar, editar y ensamblar el código ensamblador a simular. Los programas se guardan con la extensión *asm*. Los errores se mostrarán automáticamente durante la edición del programa.

El entorno tiene soporte para gestionar puntos de interrupción (*breakpoints*) durante la simulación.

B. *Ventana Memoria*

Esta ventana representa el contenido de la memoria en hexadecimal y permite agrupar las posiciones de memoria formando bytes, medias palabra, palabras o dobles palabras. Tanto el contenido del segmento de código como el del segmento de datos puede ser también modificado directamente por el usuario.

Cuando se realiza una escritura en memoria, bien sea por el usuario o por el procesador, la ventana muestra en rojo el último dato que ha sido modificado.

C. *Ventana Instrucciones*

Lista las instrucciones que se encuentran almacenadas en el segmento de código y las decodifica. Para cada instrucción de la lista se especifican los siguientes campos: Dirección de memoria donde comienza la instrucción, Codificación de la instrucción expresada en hexadecimal, Nemónico de la instrucción, Texto de la instrucción según aparece en el editor del código de usuario, y Etapa en la que se encuentra cada instrucción durante la simulación.

Esta ventana permite además editar las instrucciones y establecer puntos de interrupción durante la ejecución.

D. *Ventana Pipeline*

La arquitectura MIPS32 dispone de varias unidades funcionales de cálculo, unas integradas en el procesador y otras alojadas en un coprocesador independiente, cada una de ellas dedicada a realizar ciertos tipos de operaciones. Las unidades funcionales que posee el MIPS32 y que han sido implementadas en el simulador son: Unidad de enteros principal (*EXi*), Unidad para multiplicación de enteros (*EXm*), Unidad para la división de enteros (*EXd*), Unidad de FP principal (*EXfp*), Unidad para la multiplicación en coma flotante (*EXfpm*), y Unidad para la división en coma flotante (*EXfpd*).

Salvo la unidad funcional *EXi*, estas unidades realizan operaciones complejas por lo que suelen precisar de varios ciclos de reloj para realizar el cálculo, así que la etapa EX de tales instrucciones dura más de un ciclo. Además, algunas de estas unidades de cálculo multiciclo pueden estar a su vez segmentadas, esto es, que el cálculo de la operación puede estar dividido en varias subetapas. La ventaja de las unidades segmentadas es que permiten comenzar una nueva operación en cada ciclo de reloj, aunque otras anteriores no hayan terminado.

Un ejemplo de la ventana Pipeline se encuentra en la Fig. 5. En ella se representa el estado del pipeline mediante cajas. Para especificar el estado de cada etapa, el borde del recuadro que representa a la etapa se colorea según el resultado de la ejecución de la instrucción alojada en ella. Un borde de color negro especifica que la ejecución se ha realizado con éxito y otro color representa el tipo del bloqueo según la configuración de colores especificada en la configuración del simulador.

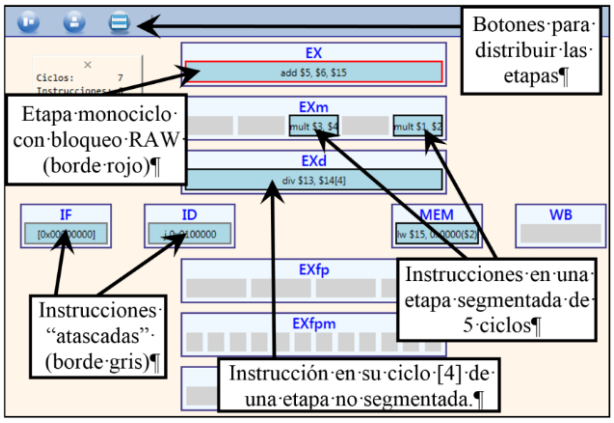


Fig. 5. Ventana Pipeline.

Cada caja que representa a una etapa se puede mover y cambiar de tamaño para adaptarse a las necesidades del usuario. También dispone de botones que simplifican la distribución de las etapas en el espacio de trabajo y reorganizan las subetapas de cada etapa horizontal o verticalmente.

E. Ventana Cronograma

Una de las ventanas que aporta más información al simular el procesamiento segmentado es la ventana Cronograma. Como puede apreciarse en la Fig. 6, en esta ventana se representa el diagrama de tiempos del estado de cada etapa en cada ciclo de reloj durante la ejecución de las instrucciones del código en ensamblador.

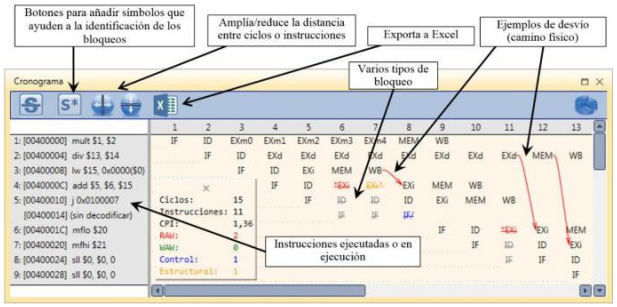


Fig. 6. Ventana Cronograma.

Como suele ser lo habitual, el eje de ordenadas indica las instrucciones y las abscisas el ciclo de ejecución. Para cada instrucción se especifica un número de secuencia, que ocupa en la ejecución de la instrucción, la dirección de memoria y su representación en ensamblador. Para cada intersección instrucción-ciclo se muestra el nombre de la etapa y el estado en el que se encuentra. Cuando la etapa ha sido bloqueada se utiliza un código de colores configurable que identifica el tipo de bloqueo. Para una mejor identificación del bloqueo, es posible opcionalmente tachar el nombre de la etapa y añadir un carácter especial junto al nombre. El significado de estos caracteres especiales es el siguiente:

- Asterisco como prefijo (*): denota que la etapa posee un bloqueo tipo lectura tras escritura (*RAW*). Por defecto se representan en rojo. Por ejemplo, ~~*EXi~~.
- Asterisco como sufijo (*): especifica que la etapa posee un bloqueo tipo escritura tras escritura (WAW) y se representan en verde. Por ejemplo, ~~EX*~~.
- Acento circunflejo (^): representa a los bloqueos estructurales y suelen aparecer en color amarillo. Por ejemplo, ~~EX^~~.
- Barra ascendente (/). Indica un bloqueo de control y es dibujado en color azul. Por ejemplo. ~~IF/~~.

Cuando la instrucción no puede avanzar de etapa debido a la detención del cauce no se utiliza ningún carácter especial, tan sólo se muestra la etapa en gris por defecto y aparece tachada.

En una etapa multiciclo, la instrucción deberá permanecer en la unidad funcional durante varios ciclos. Si la etapa está segmentada, el cronograma muestra junto al nombre de la etapa el número de la subetapa en la que se encuentra la instrucción. Véase como ejemplo la instrucción 1 que aparece en la Fig. 6; en ella puede verse que la instrucción *mult* realizó su fase EX en la unidad de multiplicación de enteros *EXm*, que está segmentada y que dura 5 ciclos. Dicha instrucción accedió a la subetapa 0 de *EXm* en el ciclo 3 y fue avanzando a las siguientes subetapas hasta alcanzar la última subetapa en el ciclo 7. En cambio, las etapas configuradas como multiciclo no segmentadas no añaden ningún identificador de la subetapa en la que se encuentran ya que en realidad la subetapa es única, sólo que dura varios ciclos. La Fig. 6 muestra un ejemplo de ello en la instrucción 2, la cual entra en la subetapa que posee *EXd* en el ciclo 4 y permanece en ella hasta pasar a la etapa de *MEM*.

Llegados a este punto, queda descrito en profundidad el simulador implementado. A continuación, se describirá el mecanismo utilizado para su evaluación y los resultados obtenidos.

IV. EVALUACIÓN

En la asignatura Arquitectura de Computadores, en la cual participan los autores de este trabajo, se estudia la arquitectura *Von Neumann* al completo, haciendo hincapié en la ejecución de instrucciones en el procesador. Puesto que la implementación de este simulador se llevó a cabo para satisfacer las necesidades docentes de esta asignatura, las funcionalidades que integra tienen relación directa con el temario impartido en ésta.

La parte del procesador de esta asignatura engloba casi dos meses de docencia presencial: seis sesiones teóricas de dos horas y cinco sesiones prácticas de dos horas. Lo que hace un total de 22 horas de las 60 que constituyen la asignatura. Aunque el simulador está a libre disposición del alumnado para que pueda trabajar con él desde casa e, incluso, utilizarlo para resolver ejercicios realizados en las sesiones teóricas, éste se utiliza de manera activa en las sesiones prácticas indicadas anteriormente.

En ellas, el alumno realiza una serie de ejercicios entregables utilizando el simulador para comprender el funcionamiento del paralelismo a nivel de instrucciones.

Descrito resumidamente el entorno en el que se utiliza el simulador, indicamos a continuación las métricas que se tienen en cuenta para evaluar la mejoría en el aprendizaje mediante el uso del simulador. Todas ellas, evalúan datos obtenidos durante 4 cursos académicos

consecutivos y aplicadas sobre una población que abarca 12 subgrupos prácticos en dos titulaciones diferentes (una media de más de 200 alumnos por año). Estas métricas son:

- Las calificaciones en las sesiones prácticas.
- Asistencia a las sesiones prácticas correspondientes.
- Encuestas de satisfacción del alumnado

A. Evolución de las calificaciones en las sesiones prácticas.

Hasta el curso académico 2014-15 se hacía uso de otro simulador que ayudaba en la explicación de una parte del contenido teórico. Esta herramienta software era muy limitada y por tanto no cubría todos los aspectos estudiados del procesador, por lo que era necesario complementarlo con un segundo simulador. El nuevo simulador integra todos los contenidos que el alumno debe estudiar, lo que facilita la comprensión del funcionamiento global del procesador al poder examinar todos los detalles de la simulación en un solo vistazo. La Fig. 7 muestra una gráfica con la evolución de las notas media obtenidas por los alumnos desde el año académico 2011-2012, 4 años antes de utilizar el nuevo simulador en las sesiones prácticas.

Fig. 7. Evolución de las calificaciones de prácticas.

Obteniendo la media de los cursos anteriores a la implantación de la herramienta VisualMIPS se obtiene una calificación general de 5.43, significativamente inferior a la media de los cuatro cursos donde se utilizó la herramienta (6.28).

B. Evolución de la Asistencia a las sesiones prácticas.

La asistencia a las sesiones de prácticas es un factor indispensable en este estudio, pues permite cuantificar el grado de motivación del alumnado y su capacidad para mantener su atención en la asignatura al trabajar con un simulador con una interfaz más atractiva.

Desde la implantación de los nuevos grados de Informática, existía un porcentaje preocupante de abandono en la asignatura, a pesar de que las prácticas eran obligatorias. Como se aprecia en la gráfica de la figura 8, antes de la utilización de VisualMips, la tasa de abandono era alta y con tendencia a aumentar cada año.

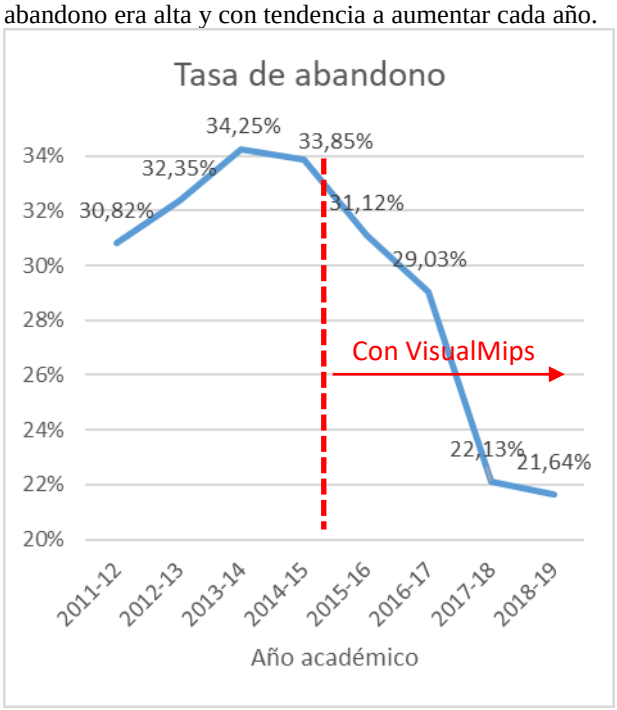


Fig. 8. Evolución del porcentaje de asistencia a prácticas.

Desde el año en el que se empezó a utilizar el nuevo simulador, la tasa de abandono ha descendido paulatinamente, a pesar de que la calificación de prácticas tiene un peso en la evaluación de la asignatura inferior que años anteriores.

C. Encuestas de satisfacción del alumnado.

Otro factor a tener en cuenta para cuantificar la usabilidad, la utilidad y la experiencia del usuario con la herramienta es la opinión de los usuarios que la usan. De manera subjetiva y anónima los alumnos realizan una encuesta personal en la que evalúan ciertos aspectos y aportan comentarios personales acerca de la herramienta. Para ello, al final de la última sesión práctica de procesadores, el alumnado rellenó una encuesta de 8 preguntas evaluadas de 1 a 8 cuyo enunciado son los siguientes:

1. Me ha ayudado a solventar dudas que poseía en las clases teóricas.
2. La notación de la que hace uso es intuitiva y está relacionada al 100% con el contenido teórico.
3. Posee todos los aspectos necesarios para comprender el funcionamiento del procesador (no le falta nada).
4. Sería interesante ampliar el simulador con módulos para acceso a memoria y entrada/salida.
5. El diseño de la aplicación es usable y fácilmente accesible para una persona que lo utilice por primera vez.
6. He hecho uso del simulador para comprobar los resultados de ciertos ejercicios teóricos.

7. El simulador utilizado me ha ayudado a comprender claramente el funcionamiento de la ejecución en pipeline.
8. En general, me parece una herramienta muy útil y completa.

La encuesta, al ser voluntaria, ha sido realizada por 133 alumnos entre los cursos 15-16, 16-17 y 17-18 (en este curso no se ha realizado aún) y se obtuvo la siguiente calificación en cada una de las preguntas:

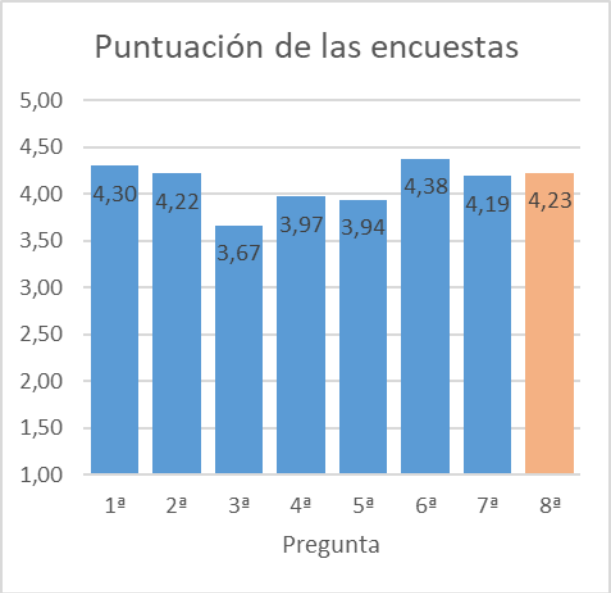


Fig.9. Resultado de las encuestas de los alumnos.

Como se aprecia en la gráfica de la figura 9, el grado de satisfacción del alumnado con la herramienta es alta, tanto en su efectividad para el aprendizaje (preguntas 1, 2, 3 y 7), como herramienta de apoyo a los ejercicios de clase (pregunta 6), o respecto a su usabilidad, claridad y sencillez (pregunta 5). De forma global obtiene una calificación de 4,23 sobre 5, por lo que se convierte en una herramienta idónea para el aprendizaje.

Además de las 8 preguntas evaluables, la encuesta incluye un apartado de observaciones para que el alumno pudiese expresar algún aspecto a destacar, sobre el que quiera incidir o que desee criticar de la herramienta. La mayoría de las observaciones de los alumnos van encaminadas a posibles mejoras o para indicar algunos errores que fueron detectados. Todos los errores y muchas de las mejoras fueron tenidas en cuenta en posteriores versiones.

V. MEJORAS FUTURAS

Desde el año 2015, en el que se implementó la primera versión del simulador, el programa está en constante evolución, corrigiéndose posibles errores, mejorando la interfaz gráfica, ampliando el repertorio de instrucciones, añadiendo nuevos elementos que forman parte del computador.

Las últimas versiones del simulador añaden la jerarquía de la memoria en fase beta para ser integrado al proyecto educativo en los próximos años.

VI. CONCLUSIONES

En este trabajo se presenta una herramienta software para facilitar el aprendizaje del paralelismo a nivel de instrucciones (ILP) en un procesador segmentado con juego de instrucciones RISC.

La herramienta es completamente configurable y ha demostrado, mediante encuestas y resultados, ser de gran utilidad en la enseñanza de Arquitectura de Computadores en los grados de Ingeniería Informática.

Los resultados presentados demuestran la reducción significativa de la tasa de abandono, así como la mejora de los resultados ligados a la parte en la que se centra la herramienta.

Es, además, una herramienta en continuo desarrollo y en uso en los diversos grados de Ingeniería Informática de la Universidad de Sevilla.

AGRADECIMIENTOS

Este trabajo ha sido desarrollado y financiado dentro del grupo de investigación TEP-108: Robótica y Tecnología de Computadores de la Universidad de Sevilla.

REFERENCIAS

[1] D. Patterson and J. Hennessy, “Computer Organization and Design : The Hardware / Software Interface,” *Comput. Organ. Des. Hardw. / Softw. Interface*, 2014.

[2] J. L. Hennessy and D. a Patterson, *Computer Architecture, Fourth Edition: A Quantitative Approach*. 2007.

[3] J. L. S. C. Pereira, “Educational package based on the MIPS architecture for FPGA platforms,” Faculdade de Engenharia da Universidade do Porto, 2009.

[4] A. Clements, “Undergraduate curriculum in computer architecture,” *IEEE Micro*, 2000.

[5] J. Djordjevic, B. Nikolic, B. Tanja, and A. MilenkoviĆ, “Cal2: Computer aided learning in computer architecture laboratory,” *Comput. Appl. Eng. Educ.*, 2008.

[6] B. Nikolic, Z. Radivojevic, J. Djordjevic, and V. Milutinovic, “A survey and evaluation of simulators suitable for teaching courses in computer architecture and organization,” *IEEE Transactions on Education*. 2009.

[7] H. Oztekin, F. Temurtas, and A. Gulbag, “BZK.SAU: Implementing a hardware and software-based computer architecture simulator for educational purpose,” in *2010 International Conference on Computer Design and Applications, ICCDA 2010*, 2010.

[8] B. Mustafa, “Modern Computer Architecture Teaching and Learning Support : An Experience in Evaluation,” in *Information Society (i-Society), 2011 International Conference on*, 2011.

[9] “MIPS32® Instruction Set Quick Reference, Revision 1.01.” [Online]. Available: http://www.imgtec.com/downloads/factsheets/MD00565-2B-MIPS32-QRC01.01.pdf.

[10] “MIPS® Architecture for Programmers Volume I-A: Introduction to the MIPS32® Architecture. Revision v6.01.” [Online]. Available: http://www.imgtec.com/mips/architec%0Atures/mips32.asp.

[11] “MIPS® Architecture for Programmers Volume II-A: The MIPS32® Instruction Set, Revision 5.04.” [Online]. Available: http://www.imgtec.com/mips/architectures/mip%0As32.asp.

Diseño, configuración y evaluación de cluster de Raspberry pi para el procesamiento paralelo de vídeo

Luis Muñoz Saavedra [1] , Lourdes Miró Amarante [1] y Manuel J. Domínguez Morales [1]

Resumen— **En este trabajo se evalúa el uso de un clúster local, basado en Raspberry Pi, para el procesado paralelo de imágenes provenientes de un vídeo. Este clúster está configurado en una red tipo bus en la que el dispositivo central es un elemento de conmutación (switch) al que se conectan todos los nodos mediante red cableada (ethernet). La sincronización se lleva a cabo mediante un protocolo asíncrono maestro-esclavo en la que una Raspberry Pi funciona como maestro en todas las transacciones utilizando un lenguaje de comunicación basado en MPI (message passing interface). El nodo maestro, disecciona el vídeo a procesar y reparte sus frames entre el resto de nodos que conforman el clúster; cada nodo trabaja independientemente aplicando el filtrado correspondiente sobre el frame que posee y retorna los resultados al nodos maestro una vez finalizado el procedimiento.**

Para evaluar la bondad del clúster, se hace uso de un banco de pruebas extenso con diversos tamaños y cantidad de imágenes, controlando los tiempos de transferencia y procesado para cada uno de los casos. Los resultados se obtienen en base a la aceleración obtenida para cada caso, comparándolo con su versión secuencial. Se incluye, además, una comparativa con otros equipos en la que se muestra la mejoría obtenida por el clúster.

Palabras clave—**Paralelismo, clúster, MPI, OpenCV, multithreading, Raspberry pi .**

I. Introducción

En la actualidad, las técnicas de procesamiento de vídeo que se están aplicando requieren cada vez de más capacidad de cómputo: casos como Big Data o Deep Learning son prueba de ello. Estas técnicas requieren de gran cantidad de procesamiento de datos para poder funcionar de manera precisa y, para ello, es necesario hacer uso de sistemas de altas prestaciones. Actualmente, la solución aportada pasa por utilizar sistemas distribuidos (ya sea locales o en la nube) para suplir las carencias de un único equipo local.

Entidades como la Unión Europa han visto en este campo un punto crítico para no perder el competitividad con el resto del mundo y han situado la supercomputación como objetivo [1], buscando tener el primer ordenador del orden del exa para el año 2026. Para que la Unión Europea cumpla sus objetivos es necesario aportar capital humano, y es necesario formar técnicos especializados en el campo de la supercomputación: será un error tener grandes máquinas y no tener la gente debidamente cualificada para poder exprimir al máximo los equipos con los que estén trabajando.

[1]Departamento de Arquitectura y Tecnología de Computadores (Universidad de Sevilla). E-mail: luimunsaa@atc.us.es

Sin embargo, esta vertiente conlleva tres problemas importantes: en primera instancia, la potencia eléctrica requerida para alimentar este tipo de sistemas [2], [3]; como segunda contraposición, la portabilidad de los mismos; y, por último, el coste necesario para montar un sistema de estas características [4].

Es, por tanto, necesario ahondar en determinadas alternativas que sean más portables y necesiten menor potencia eléctrica, aunque ello conlleve una reducción parcial de su capacidad computacional. De igual forma, y siguiendo la vertiente de la Unión Europea, es importante comenzar con la adquisición de los conocimientos necesarios para utilizar los sistemas de supercomputación [5]. Todo ello se aúna en este proyecto, el en cual se realiza el diseño e implementación de un clúster de bajo coste para su uso en tareas de procesamiento con altas necesidades de cómputo. El resultado del trabajo se puede comparar en cuanto a aceleración, para justificar la conveniencia de hacer uso del clúster, y con otros equipos de mayores prestaciones. Por otro lado, y finalizando la justificación del trabajo, aunque el dispositivo que se expone esté lejos en cuanto a las prestaciones que está buscando la Unión Europa, es un proyecto que tiene una alta utilidad docente, ya que con éste se puede introducir al mundo de la supercomputación a personas que no conocían este campo de la informática [6].

Existen otros trabajos en los que se detalla el montaje y funcionamiento de un clúster basado en nodos Raspberry Pi [7], sin embargo se limitan a exponer el montaje y configuración sin llegar a aplicarlo a un caso real [8]. En este trabajo, no se busca exclusivamente ese fin, sino también comprobar la utilidad y eficiencia del mismo en problemas reales, evaluando los beneficios que ello conlleva [9].

El resto del manuscrito se organiza de la siguiente forma: en la sección 2 se describe la arquitectura hardware del sistema implementado; a continuación, se ahonda en las diversas implementaciones software de los algoritmos de procesado; en la sección 4 se muestran los resultados obtenidos con las diversas pruebas realizadas; y, finalmente, se exponen las conclusiones del trabajo.

II. Descripción del sistema

La arquitectura hardware del sistema está compuesta por seis sistemas empotrados Raspberry Pi formando una red tipo bus. En ella, todos los nodos están conectados a un elemento central de con-

mutación (switch) para compartir la red de comunicación. Aun así, la comunicación se centraliza a través de un nodo maestro, que es el que realiza el procesado y reparto inicial entre el resto de nodos del clúster.

El usuario se comunica directamente con el nodo maestro, enviando el archivo de vídeo que se pretende procesar. éste, haciendo uso de la librería OpenCV, realiza una división frame a frame y reparte el trabajo entre los diversos nodos mediante una comunicación por paso de mensajes (MPI). Una vez ha realizado el reparto, el nodo maestro no se queda ocioso sino que también interviene en dicho procesado. Cada nodo posee una implementación más sencilla que la que integra el nodo maestro, exclusivamente centrados en la recepción de frames, procesado de las mismas mediante OpenCV y devolución de resultados por paso de mensajes. Una vez que el nodo maestro recibe todos los frames procesados, continua con los siguientes hasta finalizar el vídeo. La arquitectura implementada puede observarse en la Figura 1.

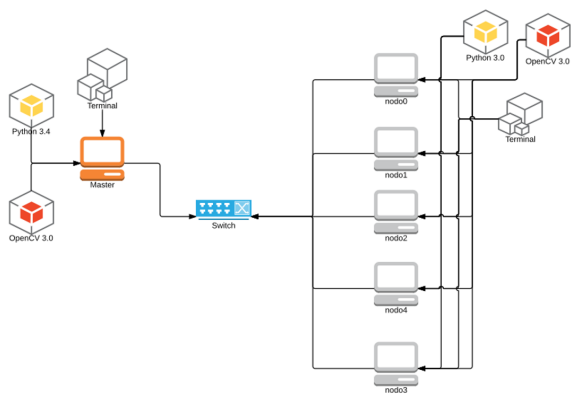


Fig. 1. Arquitectura hardware del sistema.

Como se comentó anteriormente, cada nodo consiste en una Raspberry Pi: en un trabajo previo, se llevó a cabo la implementación utilizando Raspberry Pi 2; sin embargo, en esta ocasión se hace uso de seis Raspberry Pi 3 B. Se ha seleccionado finalmente este modelo ya que posee mayor capacidad de cómputo que el modelo anterior: como se puede observar en la Tabla I, la Raspberry Pi 3 B obtiene una aceleración de 3.5 aproximadamente (en torno a un 250% más rápida) respecto a la Raspberry Pi 2 a la hora de procesar las imágenes. Para hacer esta comparativa, se ha ejecutado un algoritmo de procesamiento sencillo en ambas placas, utilizando varios valores en cuanto al número de imágenes, y utilizando un tamaño de imagen fijo de 1 MegaByte. Además de lo anterior, han sido necesarias 6 tarjetas microSD de 16 GigaBytes, 6 cargadores de, al menos, 2.1 amperios y un switch para interconectar todas las placas Raspberry Pi.

Todo el sistema ha sido montado sobre unas estructuras superpuestas impresas en 3D (puede observarse en la Figura 2, para facilitar su portabilidad y beneficiar su refrigeración: al comienzo, estos problemas fueron continuos por el uso de carcasas cerradas para cada placa, ocasionando graves problemas de temperatura que ocasionaban reinicios esporádicos y reducción de frecuencia de reloj.

TABLA I
Comparación RBpi2 vs RBpi3B: tiempos de procesado (milisegundos) y aceleración

Imágenes	RBpi2	RBpi3B	Acel.	Mejora
1	10.54	2.82	3,73	+257%
6	61.84	17.05	3.62	+262%
12	123.48	34.44	3.58	+258%
24	246.51	70.77	3.48	+248%
48	492.04	142.68	3.44	+244%
96	987.31	285.16	3.46	+246%
192	1974.80	568.76	3.47	+247%
196	2015.69	578.23	3.48	+248%

Fig. 2. Clúster montado.

Para la interconexión de las placas hemos usado un switch, en este caso ha sido un D-link DES-1016D de 16 puertos fast ethernet, no hemos necesitado un switch superior ya que las Raspberry pi solo trabajan en fast ethernet , también ha sido necesario configurar la opciones de red de las Raspberry pi, para ello hemos fijado la dirección ip de cada una de ellas, a continuación hemos añadido en el fichero hosts de cada una de ellas las direcciones IP del resto de las Raspberry Pi con el alias que le hemos asignado a cada Raspberry Pi, desde "master" hasta "nodo4"; de este modo podemos acceder a cada Raspberry Pi con facilidad. Cada una de las seis Raspberry Pi tiene casi la misma configuración solo cambiamos el nombre que tiene asignado y la ip, con esto ganamos facilidad a la hora de escalar nuestro clúster: solo hay que realizar unos pequeños cambios que no requieren más de 5 minutos. Como las Raspberrys tienen casi la misma configuración para instalar las diferentes herramientas que usamos para analizar las imágenes, hemos usado una Raspberry Pi en la que hemos realizado todas las instalaciones. El proceso de configuración seguido es el siguiente:

- Instalamos la última versión del sistema operativo propio de Raspberry (Raspbian 9, versión simplificada de Debian) [10].
- Instalamos la última versión de Python (3.5) [11].
- Descargamos la versión 4.0.1 de OpenCV [12] y la compilamos para nuestra placa: proceso tedioso de más de 1 hora, en el que se necesita ampliar la partición swap para evitar errores.
- Instalamos la herramienta de MPI MPICH en la versión 3.3 [13].

Para la siguientes Raspberry Pi hicimos copias de la tarjeta microSD, grabándola en las restantes tarjetas; proceso que solo requiere de 40 minutos a diferencia del procedimiento original, y de esta forma disponemos 6 sistemas idénticos. Para que la herramienta de MPI funcione de manera transparente ha sido necesario generar las diferentes claves públicas y transferirlas en el sistema, en nuestro caso la placa máster puede acceder de manera transparente al resto de nodos.

Seguidamente se expondrán los algoritmos de procesado utilizados para el clúster.

III. Procesamiento

Esta sección se dividirá en dos sub-secciones: por un lado se analizarán las tres versiones del algoritmo de procesamiento implementadas; y, por último, se describirá el banco de pruebas utilizado para los resultados.

A. Algoritmos

Para realizar las diversas pruebas, se han implementado tres versiones del algoritmo de procesamiento: una versión secuencial, una versión paralela y una versión para el clúster. El procesamiento a realizar sobre cada una de las imágenes consiste en una detección de rostros sobre cada uno de los frames, con un pintado posterior del marco alrededor de los rostros detectados. Las características de las diversas versiones son expuestas a continuación:

1. Versión secuencial: implementación del procesamiento exclusivamente en el nodo maestro. Esta versión es la más sencilla e incluye, exclusivamente, el tratamiento de las imágenes con OpenCV. Esta versión sirve de base sobre la que poder comparar la mejora producida por las versiones siguientes.
2. Versión paralela: implementación del procesamiento exclusivamente en el nodo maestro, pero incluyendo librerías de ejecución paralela para poder aprovechar los hilos de ejecución de los que dispone la propia Raspberry Pi 3B. Este modo no producía mejora alguna en la Raspberry Pi 2, ya que ésta solo cuenta con un hilo de ejecución. Las librerías de ejecución paralela proveen de dos métodos diferentes para ejecutar el multiprocesacimiento. Por un lado tenemos la librería Pool de ejecución y por otro lado tenemos la librería Process, la diferencia más notable entre ellas es que con la librería Pool hay que definir el nº de CPUs que queremos usar, por defecto hemos usado todas las disponibles. Para ver que método es mejor para resolver nuestro problema, hemos hecho una serie de pruebas comparando la eficiencia en tiempo de cada librería, comparadas con el tiempo en secuencial. Los datos se muestran en la figura 3.

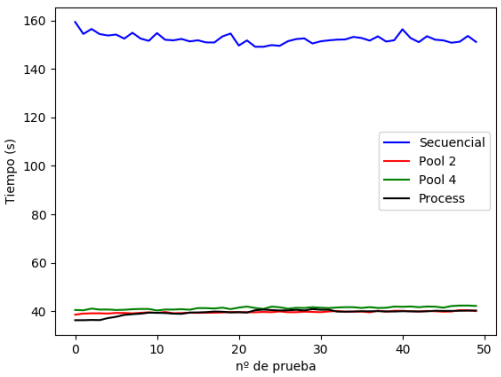


Fig. 3. Visión general de los tiempos.

Como podemos ver cualquier método mejora mucho la eficiencia del procesado en secuencial. En la figura 4 podemos apreciar con más detalle los valores obtenidos

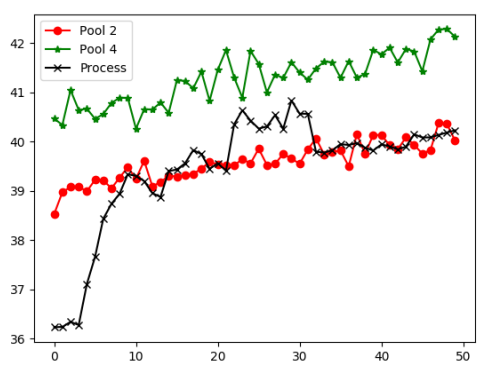


Fig. 4. Detalles de los tiempo de las librerías.

TABLA II
Tiempos medios de cada librería

Librería	T.medio(s)
Secuencial	152.4806
Pool 2	39.5807
Pool 4	41.2758
Process	39.4426

Como podemos ver en la Tabla II, al usar un Pool con 4 cpus, los tiempos de procesamiento son un 4.2% mayores, por tanto este método ha sido descartado. Por otro lado, los tiempos entre un Pool con 2 cpus y Process son muy similares

y aunque el método Process es un 0.35% más rápido que el Pool de 2 CPUs, nos hemos decantado por esta opción ya que es más estable en el tiempo.

3. Versión clúster: implementación del procesamiento en todo el clúster, incluyendo las librerías de OpenCV para tratamiento local, la librería de ejecución paralela para aprovechamiento de los hilos de ejecución locales y, finalmente, uso de la librería de MPI para comunicación por mensajes dentro del clúster y poder así aprovechar la potencia de procesamiento que aporta éste. En nuestro caso el algoritmo que hemos usado es el mismo que hemos desarrollado en la versión paralela.

Puede observarse el algoritmo de ejecución en la Figura 5.

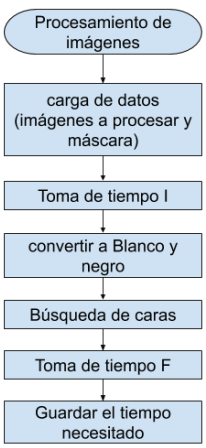


Fig. 5. Diagrama de flujo del algoritmo de procesamiento implementado.

B. Banco de pruebas

El dispositivo que hemos implementado está pensado para procesar tanto vídeo como imágenes pero, para simplificar las pruebas y focalizar los resultados en el tiempo de procesamiento, hemos supuesto que el vídeo ya se encuentra seccionado en frames, disponiendo el maestro de un directorio con las imágenes a repartir entre los nodos del clúster para que realicen su procesamiento.

Los bancos de pruebas juegan con dos aspectos importantes: tamaño de imagen y número de imágenes.

- Tamaño de imagen: en este trabajo se han utilizado tres bancos de pruebas diferentes, en los que se hacen uso de imágenes de 512 KiloBytes, 1 MegaByte y 2 MegaBytes, respectivamente. Con esta variación se pretende evaluar la capacidad de procesado local de cada nodo del clúster.
- Número de imágenes: en las pruebas, se hace uso de diverso número de imágenes total a repartir entre todos los nodos del clúster. Este número de imágenes varía entre 6 (una por nodo) y 196. Con este parámetro se pretende evaluar la incidencia de la comunicación dentro del clúster.
- Para facilitar la toma de datos se han creado una serie de scripts que automatizan el lanzamiento de las diferentes pruebas. Con esto hemos querido eliminar la necesidad de estar atentos cada vez que terminaba una prueba para poder lanzar la siguiente, con esto hemos conseguido ahorrar a la hora de lanzar las diferentes pruebas, ahorrando con esto tiempo de ejecución, ya que las pruebas se iban lanzando automáticamente.
- Para obtener unos datos más consistente y evitar en la medida de lo posible las posibles interferencias generadas por algún proceso lanzado en segundo plano por la máquina o posibles problemas a la hora del envío o recepción de los mensajes para empezar las pruebas. Para ello hemos decidido lanzar 10 veces cada prueba.
- Las pruebas que realizaremos consisten en 3 procesados diferentes, el primero es el análisis secuencial de las diferentes baterías de fotos, posteriormente se lanzarán las pruebas usando la paralelización usando un Pool de 2 CPUs y finalmente lanzaremos la ejecución en el clúster. El diagrama de flujo de estas operaciones lo podemos ver en la Figura 6.

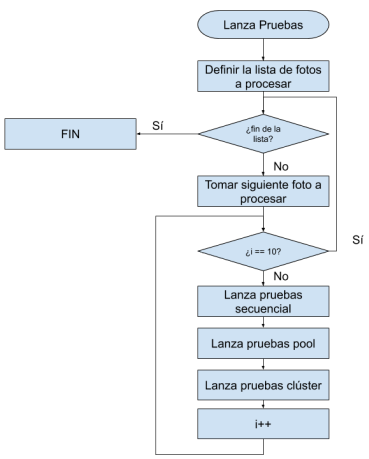


Fig. 6. Diagrama de flujo para el lanzamiento de las diferentes pruebas

IV. Resultados

A continuación se exponen los resultados obtenidos a la hora de ejecutar en el clúster descrito en la sección 2 los diversos bancos de pruebas descritos en la sección 3. Los resultados serán analizados en detalle.

Como comparativa con un sistema de procesamiento de propósito general, en cada prueba se mostrarán en paralelo los resultados de los diversos algoritmos sobre el clúster de Raspberry, así como la implementación sobre un sistema de propósito general. Las características y comparativa de estos dos sistemas pueden observarse en la Tabla III.

TABLA III
DESCRIPCIÓN DEL CLÚSTER Y EL PORTÁTIL UTILIZADOS EN LA COMPARATIVA

Caracterśtica	Clúster	Portátil
Arquitectura	ARM 64 bits	x86 64 bits
Cores	6	2
Threads	24	4
Frecuencia	1,2 Ghz	máx.3,80 GHz
L1	32 KB	128 KB
L2	512 KB	512 KB
L3	--	4 MB
RAM	6 GB	16 GB
Memoria	96 GB	512 GB
Potencia	36 w	45 w

Las Raspberrys con las que estamos trabajando tienen un microprocesador Quad Core 1.2GHz Broadcom BCM2837 64bit CPU, 1 GigaByte de memoria RAM y conexión fast ethernet (hasta 100 Megabits/s). En el microprocesador encontramos 4 núcleos Cortex-A53 con un juego de instrucciones de 64bits, cada núcleo consta de dos niveles de caché. La caché L1 es de 32 KiloBytes de memoria y la L2 es de 512 KiloBytes. Al disponer de 6 Raspberry pi, nos encontramos con un sistema de 24 núcleos, 6 GigaBytes de memoria principal y 96 GigaBytes de memoria, teniendo un consumo medio de 36 vatios en total. Por otro lado, el portátil tiene como procesador un chip I7-7560U con dos núcleos de procesamiento y 4 hilos, con 3 niveles de caché, arquitectura x86 de 64 bits y una frecuencia de reloj variable entre 2,40 Gigahercios y los 3,80 Gigahercios, con una memoria principal de 16 GigaBytes.

Teóricamente, como caso ideal anticipándonos a los resultados analizados posteriormente, se podría obtener una aceleración en la paralelización multihilo local de 4 para la Raspberry (debido a los 4 hilos que posee cada procesador); sin embargo, una versión multihilo no es equivalente a un núcleo físico y, por lo tanto, no podrá obtener una aceleración igual que éste (siendo en este caso imposible alcanzar la aceleración de 4, si bien, en casos extremos de otros procesadores, se ha alcanzado hasta una mejoría en torno al 90-95%). Por otro lado, para el clúster se podría obtener teóricamente una aceleración máxima de 24 (4 hilos x 6 nodos = 24 procesos en paralelo; pero, con la anotación anterior, esta aceleración sería levemente inferior). A continuación, vamos a exponer los resultados obtenidos, separándolos en pruebas de 512 KiloBytes, 1 Megabyte y 2 Megabytes.Para las pruebas nos vamos a centrar en los 6 siguientes tiempos:

- Tiempo secuencial RPi3: tiempo que necesita una Raspberry pi para procesar las imágenes con un solo hilo.
- Tiempo Pool RPi3: tiempo necesario para que una Raspberry pi procese las imágenes con las librerías de Pool [14].
- Tiempo clúster: tiempo que ha necesitado el clúster para procesar las imágenes.
- Tiempo secuencial portátil: tiempo necesario para que el portátil para procesar las imágenes.
- Tiempo Pool portátil: tiempo para procesar usando todos los cores del equipo.

Para cada uno de los tamaño de imágenes, se utilizan bancos de prueba con un número de imágenes variable entre 6 y 196 (como se describió anteriormente). Se expondrán en detalle dos casos (96 y 196), dejando el resto de resultados para ser presentados en las conclusiones finales.

A. Pruebas con imágenes de 512 KiloBytes

A continuación, se presentan los resultados para 96 imágenes y para 196 imágenes.

A.1 96 imágenes

Haciendo uso de 96 imágenes, el nodo maestro distribuye 16 imágenes a cada nodo del clúster. Los resultados pueden apreciarse en la Figura 7 y en la Tabla IV.

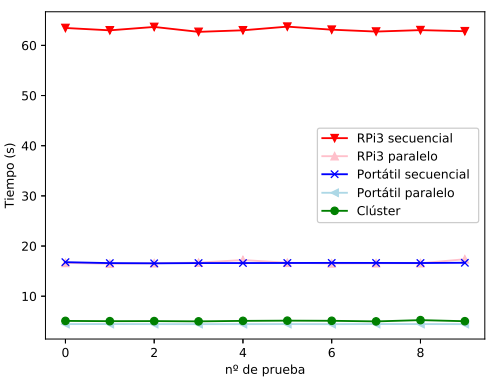


Fig. 7. Tiempo para procesar 6 imágenes de 512 KiloBytes

Puede observarse que el tiempo más lento es el ocasionado por la ejecución secuencial en la Rapsberry Pi, mientras que los más rápidos son la ejecución paralela en el portátil i7 y la ejecución en el clúster. Por lo tanto, la ejecución en un clúster formado por seis sistemas empotrados de bajo coste se asemeja en tiempo de procesamiento a un portátil de alta gama.

TABLA IV
TIEMPOS MEDIOS PARA PROCESAR 96 IMÁGENES DE 512 KILOBYTES

Prueba	T.medio(s)
Secuencial RPi	63.1329
Paralelo RPi	16.7288
Clúster RPi	5.0658
Secuencial Portátil	16.6330
Paralelo Portátil	4.4372

A.2 196 imágenes

Haciendo uso de 196 imágenes, el nodo maestro distribuye 32-33 imágenes a cada nodo del clúster.

Los resultados pueden apreciarse en la Figura 8 y en la Tabla V.

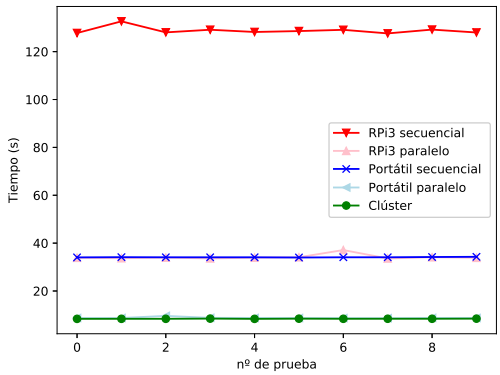


Fig. 8. Tiempo para 196 imágenes

Pueden observarse datos similares al caso anterior pero, en esta ocasión, el procesamiento en el clúster mejora al procesamiento paralelo en el portátil.

TABLA V
TIEMPOS MEDIOS DE CADA PRUEBA

Prueba	T.medio(s)
Secuencial RPi	128.8424
Paralelo RPi	34.1776
Clúster RPi	8.4022
Secuencial Portátil	34.1042
Paralelo Portátil	8.8087

B. Pruebas con imágenes de 1 MegaBytes

Las pruebas de 1 MegaByte, siguiendo el mismo procedimiento que el apartado anterior, han quedado desgranadas de las siguiente manera:

B.1 96 imágenes

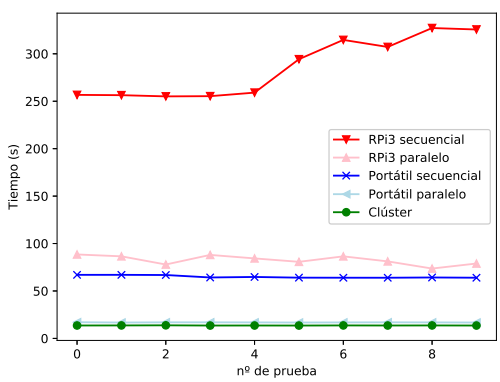


Fig. 9. Tiempo para 96 imágenes

En esta prueba, la brecha existente entre la versión del clúster y la del portátil en paralelo empiezan a distanciarse en beneficio para el clúster.

TABLA VI
TIEMPOS MEDIOS DE CADA PRUEBA

Prueba	T.medio(s)
Secuencial RPi	285.1640
Paralelo RPi	82.6224
Clúster RPi	13.6330
Secuencial Portátil	64.9961
Paralelo Portátil	16.8791

B.2 196 imágenes

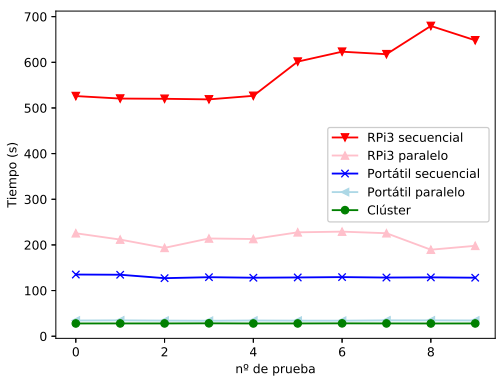


Fig. 10. Tiempo para procesar 196 imágenes de 1 MegaByte

Es importante incidir aquí, más allá del aumento de la brecha entre clúster y portátil paralelo, en el hecho de que empezamos a observar aceleraciones muy cercanas al límite teórico. Observando en la Tabla VII, la media de las pruebas realizadas en la versión secuencial de la Raspberry Pi alcanza los 578 segundos, mientras que la media de la versión del clúster se encuentra en torno a 25 segundos, lo que hace una aceleración de 23.

TABLA VII
TIEMPOS MEDIOS DE CADA PRUEBA

Prueba	T.medio(s)
Secuencial RPi	578.2312
Paralelo RPi	212.7740
Clúster RPi	25.0851
Secuencial Portátil	129.8317
Paralelo Portátil	34.4065

C. Pruebas con imágenes de 2 MegaBytes

Las pruebas de 2 MegaBytes han arrojado los siguientes valores:

C.1 96 imágenes

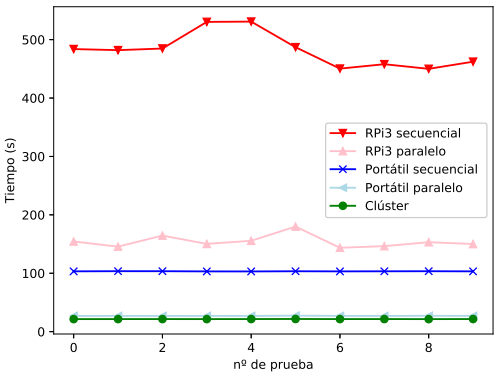


Fig. 11. Tiempo para 6 imágenes de 2 MegaBytes

Puede observarse que se sigue manteniendo la relación entre la versión clúster y la versión secuencial, con una aceleración superior a 22.

TABLA VIII
TIEMPOS MEDIOS DE CADA PRUEBA

Prueba	T.medio(s)
Secuencial RPi	481.9434
Paralelo RPi	154.3420
Clúster RPi	21.5393
Secuencial Portátil	103.3272
Paralelo Portátil	27.0354

C.2 196 imágenes

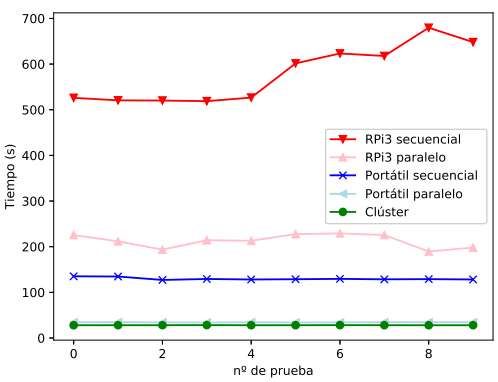


Fig. 12. Tiempo para 196 imágenes

En este último caso, se da un efecto peculiar: la relación mantenida en casos anteriores se desploma hasta bajar a una aceleración de 19. Esta circunstancia, aunque no sea analizada en profundidad, se obtiene de la capacidad de la memoria caché de las Raspberry, lo cual provoca mayor cantidad de fallos y accesos a memoria principal (repercutiendo directamente en el tiempo de ejecución).

TABLA IX
TIEMPOS MEDIOS DE CADA PRUEBA

Prueba	T.medio(s)
Secuencial RPi	878.6915
Paralelo RPi	402.5430
Clúster RPi	46.1513
Secuencial Portátil	213.1103
Paralelo Portátil	55.2972

Tras los resultados expuestos, se puede observar que la aceleración obtenida al ejecutar el algoritmo de procesamiento de vídeo en el clúster se acerca a la aceleración ideal de 24 en casos puntuales, teniendo problemas derivados con la capacidad de memoria en los dispositivos en tamaños y número de imágenes elevados.

Finalmente se expondrán las conclusiones de este trabajo.

V. CONCLUSIONES

En este trabajo se ha configurado, implementado y testeado un conjunto de máquinas Linux para poder trabajar de manera conjunta en un clúster de procesamiento de vídeo. Se han implementado diversos algoritmos de procesamiento (tanto local como distribuidos) para optimizar el tiempo de procesamiento del clúster.

A continuación, en la Figura 13, se muestra una gráfica resumen donde se puede visualizar la aceleración media obtenida para todas las combinaciones de casos evaluados: entre 6 y 196 imágenes con tamaños entre 512 KiloBytes y 2 MegaBytes.

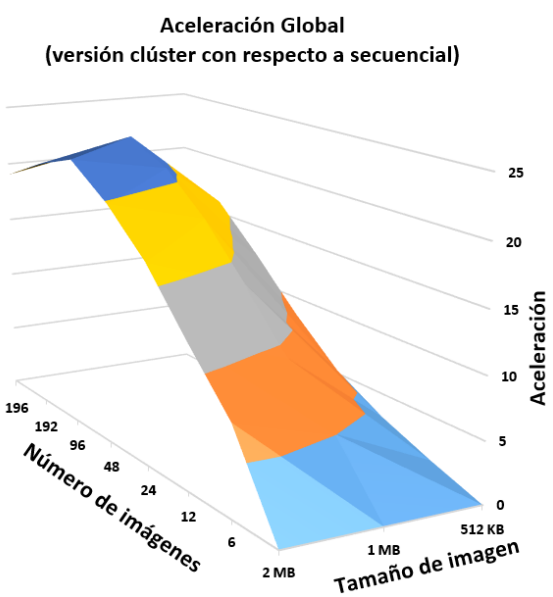


Fig. 13. Aceleración Global del clúster

Como se ha podido observar, no siempre el uso del clúster optimiza el tiempo de ejecución del procesamiento; ya que, para imágenes pequeñas, entra en juego el tiempo de transmisión, que es un factor que

no se ha tenido en cuenta y que se diluye a medida que aumenta la carga computacional del clúster.

A medida que el número de imágenes aumenta, la mejoría del clúster se hace notar frente a su versión en secuencial, obteniendo máximos cercanos al ideal (aceleración de 23 frente al máximo teórico de 24).

En cuanto a la comparativa con el portátil utilizado puede observarse que, en todo momento, la aceleración obtenida por la versión clúster del algoritmo es igual o mejor en tiempo de ejecución que la versión paralelizada del algoritmo ejecutado en el procesador i7 del portátil; esto demuestra la potencia de procesado de este tipo de dispositivos dentro de un clúster de bajo coste.

Agradecimientos

Este trabajo ha sido desarrollado y financiado dentro del grupo de investigación "TEP-108: Robótica y Tecnología de Computadores" de la Universidad de Sevilla.

Referencias

[1] Norbert Attig, Paul Gibbon, and Th Lippert, "Trends in supercomputing: The european path to exascale," *Computer Physics Communications*, vol. 182, no. 9, pp. 2041–2046, 2011.

[2] Zhonghong Ou, Bo Pang, Yang Deng, Jukka K Nurminen, Antti Yla-Jaaski, and Pan Hui, "Energy-and cost-efficiency analysis of arm-based clusters," in *Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012)*. IEEE Computer Society, 2012, pp. 115–123.

[3] Giorgio Luigi Valentini, Walter Lassonde, Samee Ullah Khan, Nasro Min-Allah, Sajjad A Madani, Juan Li, Limin Zhang, Lizhe Wang, Nasir Ghani, Joanna Kolodziej, et al., "An overview of energy efficiency techniques in cluster computing systems," *Cluster Computing*, vol. 16, no. 1, pp. 3–15, 2013.

[4] Marcos Dias De Assunção, Alexandre Di Costanzo, and Rajkumar Buyya, "Evaluating the cost-benefit of using cloud computing to extend the capacity of clusters," in *Proceedings of the 18th ACM international symposium on High performance distributed computing*. ACM, 2009, pp. 141–150.

[5] Kevin Doucet and Jian Zhang, "Learning cluster computing by creating a raspberry pi cluster," in *Proceedings of the SouthEast Conference*. ACM, 2017, pp. 191–194.

[6] Sébastien Varrette, Pascal Bouvry, Hyacinthe Cartiaux, and Fotis Georgatos, "Management of an academic hpc cluster: The ul experience," in *2014 International Conference on High Performance Computing & Simulation (HPCS)*. IEEE, 2014, pp. 959–967.

[7] Maik Schmidt, *Raspberry Pi: a quick-start guide*, Pragmatic Bookshelf, 2014.

[8] Joshua Kiepert, "Creating a raspberry pi-based beowulf cluster," *Boise State University*, pp. 1–17, 2013.

[9] Michael Cloutier, Chad Paradis, and Vincent Weaver, "A raspberry pi cluster instrumented for fine-grained power measurement," *Electronics*, vol. 5, no. 4, pp. 61, 2016.

[10] William Harrington, *Learning Raspbian*, Packt Publishing Ltd, 2015.

[11] Travis E Oliphant, "Python for scientific computing," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 10–20, 2007.

[12] Gary Bradski and Adrian Kaehler, *Learning OpenCV: Computer vision with the OpenCV library*, " O'Reilly Media, Inc.", 2008.

[13] William D Gropp, William Gropp, Ewing Lusk, and Anthony Skjellum, *Using MPI: portable parallel programming with the message-passing interface*, vol. 1, MIT press, 1999.

[14] Navtej Singh, Lisa-Marie Browne, and Ray Butler, "Parallel astronomical data processing with python: Recipes for multicore machines," *Astronomy and Computing*, vol. 2, pp. 1–10, 2013.

Experimentación Preliminar con un Trazador de Rayos para Relacionar Niveles de Abstracción

Alejandro Valero, Darío Suárez Gracia, Rubén Gran Tejero, Luis M. Ramos, Agustín Navarro-Torres, Adolfo Muñoz, Joaquín Ezpeleta, José Luis Briz, Ana C. Murillo, Eduardo Montijano, Javier Resano, María Villarroya-Gaudó, Jesús Alastruey-Benedé, Enrique Torres, Pedro Álvarez, Pablo Ibáñez y Víctor Viñals[1]

Resumen— **Para el alumnado de Ingeniería Informática resulta de gran interés alcanzar una visión global de los diferentes niveles de abstracción que permiten entender y explotar un sistema informático, sobretodo cuando en las fronteras hardware-software intervienen conceptos complejos como el paralelismo, la concurrencia, la consistencia o la atomicidad. Sin embargo, la organización habitual del Grado de Ingeniería Informática en asignaturas tiende hacia la creación de compartimentos estancos, donde se suele trabajar con un único nivel de abstracción y se pierde el panorama general.**

Este artículo proporciona un enfoque práctico para mostrar las interacciones entre los niveles de abstracción. Esto se logra implementando múltiples componentes de un trazador de rayos paralelo desde el nivel algorítmico del trazador hasta las instrucciones atómicas necesarias para garantizar la atomicidad. Los estudiantes implementan el proyecto completo a través de laboratorios de diferentes asignaturas. Cada laboratorio se centra en un único nivel de abstracción, pero muestra a los estudiantes las interacciones con el resto de niveles. Además, este trabajo también incluye un estudio de evaluación preliminar del enfoque propuesto mediante el análisis de encuestas completadas por los estudiantes.

Palabras clave— **Encuestas, futex, innovación docente, mutex, proyecto transversal, Raspberry Pi, sistema informático.**

I. Introducción

El desarrollo de un Grado en Ingeniería Informática debe permanecer al día con la rápida evolución del campo. Este hecho exige un esfuerzo en el co-diseño de sistemas hardware-software debido a que el fin de la era de la Ley de Moore y del escalado de Dennard plantea desafíos significativos en la informática. Una mejora en las prestaciones ya no se obtiene de forma gratuita simplemente con agrupar más componentes en una área similar, mientras que el consumo de energía se convierte en un problema grave. Estas son algunas de las razones principales por las que se valoran los perfiles profesionales con una visión integrada de sistema informático. Además, esta visión global permite valorar riesgos y abordar una formación posterior (especializada o no) con mayores garantías de éxito [1]. Por otro lado, en la mayoría de Grados, cada asignatura recurre generalmente a abstracciones para diseñar y explicar los sistemas informáticos.

Las abstracciones establecen límites claros en diferentes partes de un sistema y pretenden ocultar detalles innecesarios en el contexto de un nivel de sistema dado [2]. Las abstracciones también ayudan a fortalecer el proceso de aprendizaje, debido a que hacen que los estudiantes se centren en aspectos específicos. Sin embargo, según nuestra experiencia, los estudiantes a menudo pierden la visión general deseada de un sistema informático con este enfoque. Esto puede llevar a los estudiantes a la conclusión de que algunas asignaturas son independientes y no se relacionan entre sí. Muchos de ellos olvidan las implicaciones hardware que subyacen en las abstracciones de alto nivel, en términos de prestaciones y consumo.

Algunos trabajos anteriores proponen distintas abstracciones de alto nivel para facilitar el diseño de algoritmos y software [3], [4]. A diferencia de estos enfoques, este trabajo aborda el problema mencionado desde los niveles más altos a los más bajos de abstracción que subyacen en las aplicaciones paralelas y complejas de un sistema informático [5]. En concreto, este artículo expone a los estudiantes cómo la Arquitectura de Lenguaje Máquina (ALMA) y el sistema operativo proporcionan el apoyo necesario para las operaciones de sincronización de alto nivel, las cuales a su vez fortalecen el conocimiento sobre cómo los conceptos esenciales de paralelismo, concurrencia, consistencia y atomicidad se entrelazan entre ellos y con el hardware [1], [6], [7].

Para comprender mejor las relaciones entre los conceptos mencionados anteriormente, proponemos desarrollar un proyecto transversal que involucra a varias sesiones de laboratorio de diferentes asignaturas del Grado en Informática de la Universidad de Zaragoza. La propuesta consiste en un algoritmo trazador de rayos paralelo como ejemplo motivador que utiliza una cola concurrente para asignar tareas a diferentes hilos de ejecución. El acceso a la cola se realiza en exclusión mutua para preservar la integridad de los datos. Con este propósito, el acceso a la cola se administra de acuerdo con cada nivel de abstracción, con *mutex* o *futex* (*fast userspace mutex*) implementados de tres maneras distintas: funciones de biblioteca, llamadas al sistema o directamente en lenguaje ensamblador. De esta manera, el proyecto propuesto abarca los niveles de abstracción de Aplicación, Biblioteca, Sistema Operativo y ALMA, e involucra a los cursos de Informática Gráfica (IG), Programación de Sistemas Concurrentes y Distribuidos (PCD), Sistemas Operativos (SO) y Multiprocesadores (MP), respectivamente.

Cada laboratorio del proyecto se sitúa en un nivel de abstracción, pero como novedad, se añade un contexto referenciando al resto de niveles, contribuyendo de esta

[1]Dpto. de Informática e Ingenería de Sistemas, Universidad de Zaragoza, e-mails: {alvabre, dario, rgran, luisma, agusnt, adolfo, ezpeleta, briz, acm, emonti, jresano, mvg, jalastru, ktm, alvaper, imarin, victor}@unizar.es.

TABLA I: Relaciones entre niveles de abstracción, asignaturas, actividades, curso académico y semestres.

Nivel de abstracción	*Asig.*	*Actividad*	*Curso acad.*	*Semestre*
Aplicación	IG	Trazador	4^{to}	Otoño
Biblioteca	PCD	Cola de tareas	2^{do}	Otoño
Sistema Operativo	SO	Llamadas al sistema futex	2^{do}	Otoño
ALMA	MP	Futex con ensamblador	3^{ro}	Primavera

manera a integrar los diferentes niveles de abstracción. En este trabajo presentamos las pautas y objetivos principales que permiten implementar este y otros proyectos que refuercen el aprendizaje transversal. Además, para el desarrollo del proyecto utilizamos una única plataforma hardware en todos los laboratorios como es el caso de Raspberry Pi, lo cual contribuye a consolidar una visión integrada del sistema.

El proyecto presentado se desplegará en el Grado durante el próximo curso académico. Sin embargo, ya se dispone de estudios de evaluación preliminares obtenidos en el curso académico actual gracias a un grupo de estudiantes voluntarios. Este trabajo muestra los resultados experimentales para el laboratorio de SO, los cuales incluyen tanto los detalles técnicos de la sesión como los resultados de aprendizaje de los estudiantes mediante encuestas previas y posteriores. Estas encuestas revelan que los estudiantes exigen una comprensión más profunda de las interacciones entre el sistema operativo y los niveles restantes, a la vez que dichas demandas se satisfacen con la realización del laboratorio.

El resto del artículo se organiza como sigue. La Sección II describe el proyecto transversal propuesto. La Sección III muestra los resultados experimentales, diferenciando entre resultados técnicos del laboratorio y resultados de aprendizaje de los estudiantes. Finalmente, la Sección IV concluye el trabajo.

II. Propuesta de Proyecto Transversal

Esta sección presenta el proyecto conductor mediante el cual los estudiantes podrán disponer de una percepción global de un sistema informático. El material docente consiste en un enunciado de prácticas, ejemplos de código y actividades a realizar en cada nivel de abstracción. Cada enunciado se desarrolla en una sesión de dos horas de prácticas en la asignatura asociada a cada nivel. El proyecto engloba un total de ocho horas. Para más detalles se refiere al lector a [8], [9].

A. Visión General

El presente proyecto permite al alumnado consolidar los conceptos de atomicidad, consistencia, paralelismo y concurrencia presentes en un sistema informático. En este trabajo, centramos el estudio en una aplicación basada en un trazador de rayos, la cual se puede paralelizar eficientemente utilizando los conceptos anteriores. La Tabla I muestra los cuatro niveles de abstracción que constituyen el sistema, así como las asignaturas que han sido seleccionadas dentro de la titulación para afrontar el problema de forma conjunta; para cada asignatura asociada a un nivel de abstracción, la tabla muestra una breve descripción de la actividad a realizar, así como el curso académico y semestre en los cuales se llevará a cabo la actividad.

De acuerdo con la distribución cronológica de las asignaturas a lo largo de los diferentes cursos académicos, los estudiantes comenzarán el proyecto en el segundo curso. El primer laboratorio, que pertenece a la asignatura obligatoria PCD, se centra en el nivel de Biblioteca. Este laboratorio se ocupa de la implementación y administración de una cola de tareas con acceso concurrente por parte de múltiples hilos. Dicho acceso debe hacerse en exclusión mutua para evitar condiciones de carrera. Para ello, los estudiantes implementan un mutex mediante funciones de biblioteca.

El siguiente laboratorio se realiza durante el mismo curso académico y semestre, y se centra en el nivel de Sistema Operativo. En este laboratorio de carácter obligatorio, se implementa un mutex en espacio de usuario mediante llamadas al sistema operativo de tipo futex [10] y primitivas atómicas, reemplazando así a las funciones de biblioteca del nivel anterior.

El siguiente curso académico cubre el tercer laboratorio del proyecto, es decir, el nivel de ALMA, el cual se desarrolla en el curso optativo MP. En esta práctica se utilizan instrucciones de código máquina para implementar el mutex/futex, lo cual permite lograr una mayor eficiencia en el consumo de energía y las prestaciones en comparación con las funciones de la biblioteca y las llamadas al sistema.

Finalmente, en el cuarto curso, los estudiantes se centran en el nivel de Aplicación, implementando un algoritmo trazador de rayos en un laboratorio de la asignatura optativa IG. En esta actividad, las tareas a realizar en una imagen se pueden paralelizar dividiendo la imagen en regiones. Estas regiones se asignan a diferentes hilos mediante el uso de la cola de tareas concurrente. En este momento, los estudiantes pueden evaluar de manera completa el proyecto, constatando las diferencias entre los distintos mecanismos de protección de la cola según se utilicen funciones de biblioteca, llamadas al sistema o instrucciones en ensamblador.

Nótese que el desarrollo del proyecto presentado está sujeto a ciertos riesgos; por ejemplo, los estudiantes que se transfieran de una institución a otra, o los estudiantes que suspendan una asignatura o simplemente no cursen las asignaturas opcionales involucradas no completarían el proyecto. Para mitigar tales riesgos, todos los laboratorios incluyen dos partes diferenciadas. La primera parte, auto-contenida, incluye el material exclusivo del laboratorio, mientras que la segunda parte vincula el laboratorio actual con los demás. Por tanto, si un estudiante no completa un laboratorio anterior o posterior, el profesorado puede proporcionar una solución, de modo que los estudiantes pueden completar la segunda parte del laboratorio y establecer los vínculos entre niveles de abstracción.

B. Niveles de Abstracción

El sistema informático objeto de estudio se presenta en las siguientes secciones siguiendo el orden cronológico que los estudiantes experimentarán.

B.1 *Cola de Tareas con Acceso Concurrente*

Este laboratorio presenta a los estudiantes la implementación de una de las estructuras de datos concurrentes más comunes: una cola. Las colas, descritas en la asignatura de Estructuras de Datos y Algoritmos, son un mecanismo muy adecuado para la resolución colaborativa de problemas en los que varios procesos necesitan coordinarse. De esta manera, los productores y consumidores pueden usar una o más colas para compartir información y coordinarse [11]. Como en cualquier estructura de datos compartidos, para preservar la integridad de los mismos, el acceso concurrente a los datos requiere el uso de algunos mecanismos de sincronización.

Los objetivos principales de este laboratorio son los siguientes: i) implementación de una cola concurrente y limitada. Controlar el acceso simultáneo a una cola requiere considerar no sólo el acceso en exclusión mutua a los elementos, sino también la sincronización de condiciones (no existe un primer elemento en una cola vacía, o no se puede insertar ningún elemento nuevo cuando la cola está llena), ii) entender representaciones de ejecución en alto nivel como C++11 `std::thread` y iii) identificar y utilizar elementos comunes de sincronización de bajo nivel como los mutex.

Los contenidos de este laboratorio se organizan en dos partes diferenciadas. La primera parte consiste en la implementación de un tipo de cola limitada de acuerdo con la especificación observada en la asignatura de Estructuras de Datos y Algoritmos. Posteriormente, las operaciones se deben rediseñadar para considerar aspectos de sincronización: además de garantizar el acceso en exclusión mutua a la estructura de datos, las operaciones de inserción y eliminación se deben implementar como operaciones bloqueantes.

Siguiendo el enfoque propuesto para la asignatura PCD, como primera tarea, los estudiantes deben diseñar el acceso concurrente a la cola utilizando la instrucción genérica `<await B S>`, donde `B` es una guarda booleana generalmente relacionada con datos compartidos y `S` es un bloque de instrucciones. La semántica de la declaración garantiza que `S` comience su ejecución siendo `B` verdadero y que ningún estado interno de `S` sea visto por el resto de hilos. El punto de vista de alto nivel de tal declaración facilita la tarea de diseñar programas concurrentes correctos, siendo este aspecto uno de los objetivos de la asignatura.

En una clase anterior, los estudiantes han estudiado la técnica de *paso de testigo* (como se propone en [12], por ejemplo) como una forma de implementar declaraciones `<await ...>` utilizando un mutex. Para la segunda parte de este laboratorio, los estudiantes adaptarán el enfoque general estudiado al diseño del tipo de cola concurrente y limitada que proponen. Nótese que, en este nivel, el mutex es visto como el nivel de abstracción más bajo para administrar la sincronización, considerándolo como un tipo de datos abstracto. Los estudiantes desconocen cómo funciona el mutex internamente, ya sea dejando a los hilos iterar (*spin-lock*) o durmiéndolos mediante el sistema operativo (*sleep*) hasta que se les otorgue el acceso a la sección crítica. Este problema se describe de manera sucinta en el laboratorio, y los estudiantes encontrarán la respuesta implementando el tipo de datos abstracto mutex en las dos actividades siguientes: *Protección de la cola de tareas con llamadas al sistema futex* (véase la Subsección II-B.2) y *Futex con código ensamblador* (véase la Subsección II-B.3).

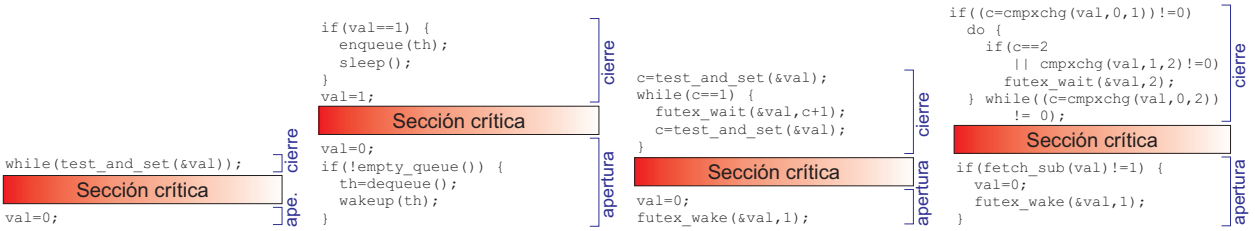
Como resultado final, los estudiantes desarrollarán dos versiones de tipo de cola concurrente y limitada. En la primera versión, cada operación en una cola debe ejecutarse en exclusión mutua. En la segunda, los estudiantes deberán adaptar el enfoque de lectores y escritores para permitir el acceso múltiple a las operaciones de *lectura* (operaciones sin efectos secundarios) al tiempo que se preserva el acceso en exclusión mutua para las operaciones de *escritura*, dando prioridad a los escritores en caso de conflicto.

Después de completar este laboratorio, los estudiantes habrán reforzado su conocimiento sobre los conceptos principales relacionados con la sincronización en sistemas concurrentes. Además, las tareas propuestas también se ocupan del uso de técnicas de diseño que se centran en la síntesis de programas concurrentes correctos.

B.2 *Protección de la Cola de Tareas con Llamadas al Sistema Futex*

Este laboratorio presenta los mecanismos requeridos por el sistema operativo para proporcionar sincronización en algoritmos concurrentes. Los objetivos principales de esta práctica son: i) mostrar el sistema operativo como proveedor de servicios para el usuario a través de llamadas al sistema, ii) aprender el uso eficiente de las llamadas al sistema futex y las primitivas de instrucciones atómicas proporcionadas por el sistema operativo y la biblioteca de lenguaje C, iii) comprender los mecanismos necesarios para proporcionar ejecución en exclusión mutua con futex e instrucciones atómicas y iv) mostrar y utilizar primitivas de cierre y apertura propias de una abstracción de mutex para gestionar el acceso a la cola de tareas concurrente implementada en la actividad previa.

El material de laboratorio describe en primer lugar las instrucciones atómicas C11 de `stdatomic.h` y solicita a los estudiantes que implementen un mutex con *spin-lock* basado en instrucciones atómicas. A continuación, se motiva la versión *sleep* de un mutex, introduciendo la intervención obligatoria del sistema operativo para cambiar el estado del hilo, y proporcionando una versión *naive* del mutex *sleep* utilizando llamadas al sistema hipotéticas como `sleep` y `wakeup`, así como operaciones de administración en una cola del sistema como `enqueue` y `dequeue`. Las limitaciones de este enfoque se utilizan para motivar las llamadas al sistema futex. A continuación, se describe la sintaxis y el uso de los parámetros de las llamadas al sistema `futex_wait` y `futex_wake`. Al utilizar estas llamadas, se guía a los estudiantes para implementar una versión intuitiva y directa de mutex *sleep* conocida como implementación básica. Finalmente, se muestra el algoritmo de pseudo-código de un mutex más eficiente como guía para codificar una implementación avanzada. Esta alternativa se basa en la implementación de mutex propuesta por U. Drepper [13], la cual está integrada en el kernel de Linux [14].



(a) *spin-lock* (b) *sleep naive* (c) *sleep* básico (d) *sleep* avanzado

Fig. 1: Procedimientos de bloqueo y apertura de los mutex *spin-lock* y *sleep*. Las versiones *sleep* incluyen llamadas al sistema para cambiar el estado de los hilos.

La Figura 1(a) muestra los procedimientos de cierre y apertura de un mutex *spin-lock* protegiendo una sección crítica. El valor de la variable `val` en espacio de usuario representa los dos estados del mutex: libre (`val=0`) y bloqueado (`val=1`). La instrucción atómica `test_and_set` cambia el estado del mutex[1]. De manera más precisa, esta instrucción establece `val` a 1 y carga su valor anterior en `c` sin la sobrecarga de una llamada al sistema. A continuación, un hilo entra en la sección crítica si el mutex está libre (`c=0`). De lo contrario, el hilo permanece iterando en el cierre. En el procedimiento de apertura, el hilo simplemente establece `val` a `0` para liberar el mutex. Puesto que el mutex *spin-lock* deja a todos los hilos en el cierre en espera activa, esta implementación sufre pérdidas significativas en las prestaciones del sistema cuando existe competencia por el mutex.

La Figura 1(b) ilustra la versión *sleep naive* de un mutex. Estos procedimientos son similares a otras versiones ofrecidas en libros de texto de conceptos de sistemas operativos como [15], [16] y [17]. Este código sólo es correcto si ambos procedimientos se ejecutan atómicamente. A diferencia de *spin-lock*, asumir una ejecución no atómica presenta varios problemas que se enumeran en el material del laboratorio y que los estudiantes deben comprender. Estos problemas son: i) las operaciones de lectura y escritura sobre `val` no se realizan de forma atómica, lo cual puede llevar a que varios hilos lean el mutex como libre, ii) las operaciones de lectura del mutex y la inserción del hilo en la cola no son atómicas, lo cual puede conducir a un hilo suspendido indefinidamente si el mutex se libera entre las operaciones de lectura e inserción y iii) después de despertarse de la llamada `sleep`, un hilo no tiene ninguna garantía de obtener mutex puesto que otro hilo puede acceder a la sección crítica antes de que el primero tome el mutex.

La Figura 1(c) muestra la implementación *sleep* básica. Esta alternativa de implementación corrije todos los comportamientos incorrectos de *sleep naive*. En la función de cierre, la instrucción atómica cambia el estado del mutex. Si el mutex está libre, el kernel no se invoca y el hilo entra en la sección crítica. De lo contrario, se invoca la llamada al sistema `futex_wait`. Esta llamada suspende al hilo que la invoca en una cola del sistema si el mutex está bloqueado (`val=1`), o devuelve inmediatamente si el mutex se ha liberado mientras tanto (`val=0`). Si el mutex está bloqueado, el hilo permanece suspendido hasta que otro hilo lo despierta. Nótese también que cada vez que `futex_wait` devuelve, el hilo intenta adquirir nuevamente el mutex.

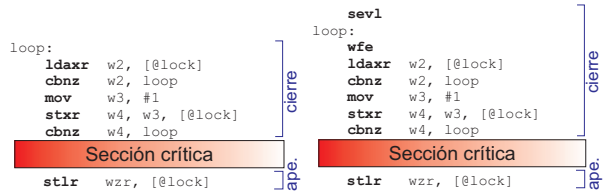
El procedimiento de apertura establece `val` a `0` y llama a `futex_wake`. Esta llamada despierta al número de hilos, entre aquellos suspendidos en la cola del sistema, indicado en el segundo argumento (`1` en el ejemplo puesto que un único hilo puede acceder a la sección crítica). Nótese que esta llamada se invoca independientemente de si el mutex está en disputa o no, lo cual puede afectar a las prestaciones del sistema.

La implementación *sleep* avanzada que se muestra en la Figura 1(d) aborda el problema de prestaciones de la versión básica. En este caso, se consideran tres estados para el mutex: libre (`val=0`), bloqueado y sin hilos esperando (`val=1`) y bloquedo y con al menos un hilo esperando (`val=2`). En el procedimiento de cierre, `test_and_set` ya no resulta útil porque `val` toma tres valores. En su lugar, se utiliza la primitiva atómica `cmpxchg`, mediante la cual se almacena un `1` (tercer argumento *deseado*) en `val` ante una comparación exitosa entre `val` y `0` (segundo argumento *esperado*). Independientemente del resultado de la comparación, el valor original de `val` se almacena en `c`. Si `c==0`, el hilo que invoca la llamada actualiza el estado del mutex a bloqueado sin hilos esperando y a continuación accede a la sección crítica. De lo contrario, el hilo se suspende en la cola del sistema llamando a `futex_wait`. Anteriormente, el segundo `cmpxchg` establece `val` a `2` si es necesario, actualizando el estado del mutex a bloqueado y al menos un hilo esperando. Nótese que, si el mutex se libera entre el primer y el segundo `cmpxchg`, este último devuelve 0 y el hilo no se suspende. El tercer `cmpxchg` asegura que un hilo toma el mutex sólo si se devuelve un 0. En tal caso, `val` se establece a `2` porque no hay certeza del número de hilos esperando en el cierre.

El método de apertura sustrae 1 a `val` con la instrucción atómica `fetch_sub`, la cual devuelve el valor anterior del argumento. La llamada `futex_wake` se invoca sólo en el caso de que haya un hilo suspendido en el cierre, evitando las costosas llamadas al sistema cuando no hay hilos suspendidos. Se remite al lector a [13] para más detalles acerca de la implementación de mutex basada en *sleep* avanzado.

Al final del laboratorio, los estudiantes utilizan las diferentes versiones de mutex para implementar una abstracción compleja, es decir, la cola de tareas concurrente utilizada en la actividad anterior. Además, también se les encomienda a evaluar la idoneidad de cada

[1]Para mayor brevedad, hemos acortado los nombres de las funciones originales de `stdatomic.h`; por ejemplo, `test_and_set` corresponde a `atomic_flag_test_and_set`, mientras que el operador de asignación para `val` se refiere a `atomic_store`.



(a) *Spin-lock* (b) Modo *sleep*

Fig. 2: Procedimientos de cierre y apertura con código ensamblador de ARMv8.

versión de mutex para diferentes escenarios de contención (véase la Sección III-A). En general, los estudiantes utilizarán llamadas al sistema futex e instrucciones atómicas para implementar las versiones *spin-lock* y *sleep* de una abstracción de sincronización básica como es el mutex.

B.3 *Implementación de Futex con Código Ensamblador*

Este laboratorio pretende ayudar a los estudiantes a comprender el apoyo proporcionado por el nivel de ALMA para implementar una exclusión mutua rápida y confiable, en términos de coherencia y atomicidad. Los procesadores ARM incluyen instrucciones *load-link/store-conditional* y *barriers*, las cuales conforman el núcleo de las estructuras de nivel superior, como los mutex y los futex. Además, estas instrucciones no requieren ningún nivel de privilegio para que se ejecuten, por lo que los programadores pueden explotarlas directamente para mejorar la eficiencia y reducir la sobrecarga de las llamadas al sistema.

Al concluir este laboratorio, los estudiantes habrán logrado los siguientes objetivos: i) entender cómo funcionan las instrucciones atómicas a nivel de ALMA para los procesadores ARMv8, ii) saber por qué a menudo se requieren *barriers* al escribir instrucciones atómicas y iii) aprender las implicaciones en las prestaciones y consumo de las diferentes implementaciones de mutex (*spin-lock* y *sleep*).

Las actividades de este laboratorio están diseñadas para ayudar a los estudiantes a lidiar con códigos complejos que mejoren sus habilidades de programación en bajo nivel, especialmente en lo que respecta a las prestaciones y la eficiencia energética. Además, estas actividades muestran lo importante que es para una ALMA proporcionar soporte para constructores complejos de alto nivel, como los mutex utilizados por los sistemas operativos, bibliotecas y aplicaciones. Finalmente, los estudiantes obtienen conocimientos sobre la relación entre el modelo de memoria C/C++11 y los modelos de consistencia correspondientes al nivel de ALMA.

El material de esta sesión está organizado en dos partes. En la primera parte, se les solicita a los estudiantes que generen una condición de carrera con la escritura de un programa multi-hilo que reduzca una matriz agregando todos los elementos sin primitivas de sincronización. A continuación, los estudiantes codifican una primitiva *fetch and add* con instrucciones *load-link* (`ldaxr`) y *store-conditional* (`stlxr`) de ARMv8 [18]. La instrucción *fetch and add* implementada se utiliza en el programa anterior para verificar que el código ahora está libre de condiciones de carrera.

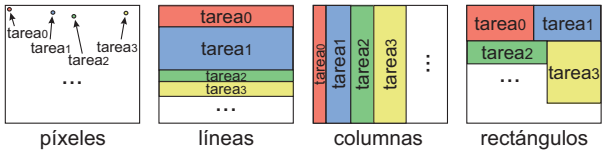
La segunda parte comprende dos tareas. La primera tarea propone una implementación básica de las funciones de cierre y apertura de un mutex basadas en las instrucciones `ldaxr/stlxr` tal y como se ilustra en la Figura 2(a). Los hilos en la función de cierre iteran hasta que adquieren el mutex. Las iteraciones ocurren entre las dos instrucciones de salto `cbnz`. Ya sea si el mutex está bloqueado (primera `cbnz`) o la instrucción `stxr` falla en el intento de bloquear el mutex (segunda `cbnz`), las instrucciones de salto devuelven el flujo del programa al comienzo del bucle. Notése también que, a diferencia de la implementación avanzada basada en llamadas al sistema futex, sólo se consideran dos estados para el mutex en este nivel, es decir, estado libre y bloqueado. Al completar esta actividad, los estudiantes descubren que ambas funciones pueden ser las primitivas de sincronización para la cola de tareas concurrente.

La segunda actividad propone una implementación avanzada de la función de cierre al reemplazar el *spin-lock*, que consume mucha energía, con una instrucción `wfe`. Esta instrucción coloca al núcleo en un estado de bajo consumo de energía sin devolver el control al sistema operativo. La Figura 2(b) muestra tal implementación de mutex *sleep*, también con dos estados de mutex, libre y bloqueado. El estudiante aprenderá cómo el sistema operativo considera que el programa se está ejecutando, mientras que en realidad está esperando que se libere el mutex, y cómo el hilo puede recuperar el mutex sin la necesidad de una llamada al sistema. En particular, la instrucción `stlr`, ubicada en la función de apertura, realiza un almacenamiento con un *barrier* de liberación y activa cualquier núcleo que pueda estar inactivo después de ejecutar la instrucción `wfe`. Para garantizar el progreso, los hilos también se activan después de una interrupción.

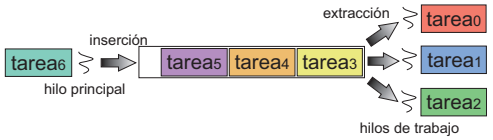
Con las implementaciones *spin-lock* y *sleep*, los estudiantes realizarán una comparación rápida entre ambas en términos de prestaciones y consumo de energía. Para este último propósito, ya que Raspberry Pi no proporciona ningún contador hardware para medir consumo, las mediciones se pueden hacer leyendo la corriente en la entrada de alimentación USB de la placa. Para realizar esta actividad, una alternativa de bajo coste consiste en un medidor de voltaje y corriente USB con pantalla. Otra opción para obtener resultados más precisos consiste en un analizador de potencia como el Newtons4th PPA500 [19]. Los analizadores de potencia admiten una tasa de muestreo mucho mayor y permiten descargar las muestras en ficheros CSV para realizar un análisis *offline*.

B.4 *Trazador de Rayos Paralelo*

La asignatura IG propone una actividad práctica que implica la implementación de un algoritmo de *path-tracing* [20], el cual se paraleliza asignando diferentes tareas a hilos de ejecución utilizando una cola de tareas concurrente. Los objetivos principales que los estudiantes lograrán mediante este laboratorio son los siguientes: i) encontrar y comprender los cuellos de botella computacionales del algoritmo, ii) diseñar estrategias de paralelización que mejoren las prestaciones sin pérdida de precisión y iii) probar, explorar y analizar el impacto (y la sobrecarga) de la combinación de diferentes estrategias



(a) Tipo de regiones de imagen



(b) Cola de tareas concurrente

Fig. 3: Diagramas de una imagen 2D dividida en tareas de renderización y una cola concurrente para asignar tareas a diferentes hilos trabajadores.

de paralelización, tipos de colas de tareas concurrentes y alternativas de mutex en las prestaciones.

El contenido de este laboratorio incluye una descripción del algoritmo de *path-tracing* y por qué puede ser paralelizado. Este algoritmo genera una imagen 2D a partir de una representación 3D de una escena virtual, incluyendo las propiedades de geometría y ópticas de los objetos, así como las caracterizaciones físicas de los sensores (cámaras) y las fuentes de luz. En la práctica, el algoritmo simula las trayectorias de los haces de luz a través de la escena virtual para obtener el color final que llega a cada uno de los píxeles de la imagen. Una parte fundamental de *path-tracing* es el trazado de rayos (*ray-tracing*), es decir, generar las trayectorias de los rayos desde una cámara. Desde el punto de vista de las prestaciones del sistema, interesa paralelizar el trazador de rayos debido a su necesidad de cómputo y al hecho de que tarda bastante tiempo en converger (aproximadamente 1 o 2 horas para obtener un resultado de buena calidad para una escena virtual simple).

Una estrategia común de paralelización del trazador de rayos es subdividir la imagen en diferentes regiones, convirtiendo el cálculo de cada una de las regiones en una tarea de renderización que se asignará a un hilo de ejecución. Los estudiantes deben explorar diferentes estrategias de paralelización en diferentes dimensiones, tal y como se ilustra en la Figura 3(a):

- Diferentes tipos de regiones de imagen: píxeles, líneas, columnas o rectángulos.
- Diferentes tamaños de región: rectángulos más pequeños o más grandes y agrupaciones de filas o columnas.

Dependiendo de la geometría y otras propiedades de la escena virtual, así como de otros detalles de implementación del algoritmo, la carga computacional puede variar enormemente de una región a otra [21]. Por esta razón, necesitamos un mecanismo seguro para distribuir tareas entre hilos. Además, dado que es imposible estimar de antemano la carga computacional de cada tarea, se requiere una cola de tareas concurrente para hilos como la que se muestra en la Figura 3(b). En esta cola, un hilo principal inserta todas las tareas de procesamiento, mientras que múltiples hilos de trabajo extraen las tareas y realizan el cómputo correspondiente.

Los estudiantes no sólo deben implementar las estrategias de paralelización, sino que también deben combinarlas con las diferentes colas de tareas de laboratorios anteriores, identificando los pros y contras de cada uno de los enfoques y analizando y justificando su impacto en las prestaciones. Por ejemplo, los estudiantes deben responder a preguntas como: *¿Cuál es el tamaño óptimo de una región? ¿Cuál de las implementaciones de cola concurrente y versión de mutex funcionan mejor y en qué circunstancias?*

En general, la implementación y la paralelización del algoritmo de *path-tracing*, junto con la evaluación de prestaciones de cada cola y mutex, ayudará a los estudiantes a comprender y analizar el efecto de los mecanismos, decisiones y detalles de implementación de bajo nivel con aplicaciones y algoritmos de alto nivel, lo cual reforzará la visión transversal de un sistema informático.

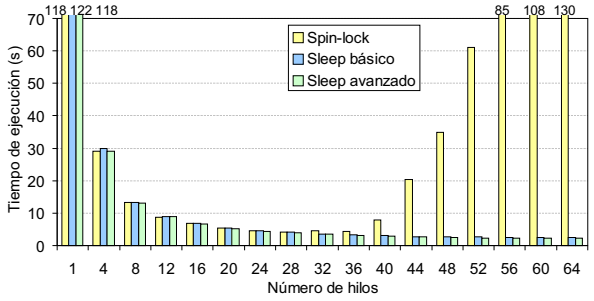
III. Resultados Experimentales

Esta sección muestra los resultados experimentales para el laboratorio de SO. En primer lugar, se muestran los resultados principales y las conclusiones que los estudiantes deben obtener con la realización de este laboratorio. De manera más precisa, se analiza el impacto en las prestaciones que obtienen las versiones de mutex basadas en *spin-lock* y *sleep*. En segundo lugar, se presenta un estudio acerca de los resultados de aprendizaje de los estudiantes.

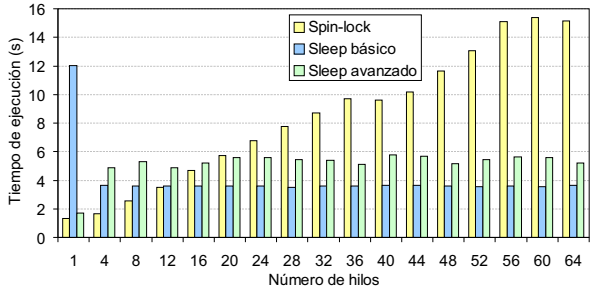
A. Impacto en las Prestaciones de las Implementaciones de Mutex

Para cuantificar el impacto en las prestaciones de diferentes alternativas de implementación de mutex, hemos medido el tiempo de ejecución (en s) de un número variable de hilos de ejecución extrayendo tareas de una cola concurrente protegida con un mutex. Se han evaluado tres mutex diferentes: *spin-lock* (Figura 1(a)), *sleep* básico (Figura 1(c)) y *sleep* avanzado (Figura 1(d)). Se consideran dos escenarios de contención diferentes, denominados real y sintético. En el escenario real, una vez que los hilos extraen de la cola una tarea, computan cierta cantidad de trabajo privado antes de volver a la función de cierre. En el escenario sintético, los hilos vuelven a la función de cierre tan pronto como liberan el mutex y sin realizar ningún trabajo adicional. Este último escenario cubre un caso extremo en el que los estudiantes pueden observar cómo un cambio en la cantidad de trabajo privado conduce a conclusiones inesperadas. La Figura 4 muestra los resultados. Todos los experimentos se ejecutan en una Raspberry Pi con una frecuencia de CPU fija de 600 MHz para garantizar la reproducibilidad de los experimentos y evitar problemas térmicos.

En el escenario real, todos los mutex analizados obtienen resultados de prestaciones muy similares para un número relativamente bajo de hilos (de 1 a 36 hilos). Con un número mayor de hilos, *spin-lock* aumenta dramáticamente la ejecución, ya que más hilos se encuentran en espera activa por conseguir el mutex. Por otro lado, los mutex *sleep* siguen reduciendo el tiempo de ejecución con el número de hilos, extrayendo



(a) Contención real



(b) Contención sintética

Fig. 4: Prestaciones de las diferentes implementaciones de mutex variando el número de hilos.

paralelismo cuando los hilos computan trabajo fuera de la sección crítica. Nótese también que *sleep* avanzado mitiga ligeramente el tiempo de ejecución con respecto a *sleep* básico. Esto se debe principalmente a que el primero no invoca costosas llamadas al sistema `futex_wake` cuando no hay hilos esperando en la función de cierre.

En el escenario sintético, en comparación con los mutex *sleep*, *spin-lock* aumenta progresivamente el tiempo de ejecución con el número de hilos debido a que todos ellos compiten por el mutex. Por el contrario, los mutex *sleep* aceleran la ejecución al despertar sólo al hilo que consigue el mutex. Nótese que no se pueden apreciar beneficios en las prestaciones de los mutex *sleep* con el número de hilos. Esto se debe a que los hilos no explotan ningún paralelismo al no computar ningún trabajo aparte de la sección crítica. Una observación interesante es que en las ejecuciones mono-hilo, *sleep* básico aumenta en gran medida el tiempo de ejecución. Esto se debe a que la función de apertura obliga a invocar a la llamada al sistema `futex_wake`, la cual produce al menos una sobrecarga por cambio de contexto, lo que a su vez puede dejar la CPU a otro proceso. La diferencia de prestaciones entre los mutex *sleep* se debe principalmente a que la versión avanzada se traduce en más instrucciones en los procedimientos de cierre y apertura. Además, el mutex *sleep* avanzado no aprovecha la llamada `futex_wake` condicionada, ya que siempre hay hilos esperando en el cierre y la llamada al sistema siempre se invoca de igual manera a como ocurre con la implementación básica.

B. Resultados de Aprendizaje de los Estudiantes

Esta sección proporciona una evaluación cualitativa y preliminar del laboratorio propuesto de SO. Este laboratorio se llevó a cabo una vez que las asignaturas SO y PCD terminaron, dando a los estudiantes la oportunidad de comparar entre el laboratorio actual (es decir, sin interacciones directas con ningún laboratorio de PCD) y el laboratorio propuesto. La asistencia a este laboratorio fue voluntaria y un 15 % de estudiantes de SO aceptó realizarlo.

Se diseñaron dos encuestas de satisfacción diferentes, denominadas encuesta previa y posterior, las cuales fueron completadas por los estudiantes antes y después de completar la sesión de laboratorio, respectivamente. Ambas encuestas suman un total de 17 preguntas y comprenden respuestas con escala de Likert (es decir, 1=*totalmente en desacuerdo*, 5=*totalmente de acuerdo*) respuestas cortas sí/no y respuestas abiertas. Todas las preguntas fueron dirigidas a medir las percepciones de los estudiantes sobre el diseño del laboratorio, su utilidad, la calidad de los materiales y recursos utilizados, así como la asistencia del profesorado durante la sesión.

Las conclusiones principales de la encuesta previa se resumen a continuación. Primero, los estudiantes consideraron que todas las asignaturas del Grado están relacionadas de alguna manera entre ellas (3.75 en promedio). En concreto, el 83 % de los estudiantes percibió que la asignatura SO se relaciona fuertemente con las áreas de arquitectura y tecnología de computadores y de computación paralela y distribuida. Sin embargo, los estudiantes dieron una puntuación de 4.15 a la necesidad de una comprensión más profunda de estas relaciones, lo que nos confirmó la necesidad de este tipo de proyectos transversales. Del conjunto de preguntas técnicas, observamos que los estudiantes perciben interacciones claras entre el sistema operativo y la ALMA, pero no tantas entre el sistema operativo y las construcciones concurrentes de alto nivel, como es el caso de los mutex de biblioteca. Este hecho confirma que los profesores de diferentes áreas deberían colaborar aún más entre sí.

La encuesta posterior reveló que la sesión de laboratorio fue bien recibida. Más de 4 de cada 5 estudiantes completaron exitosamente el laboratorio y dieron una puntuación general de 4.42 a la calidad del diseño del laboratorio, los materiales y recursos, y la asistencia del profesorado. Al finalizar el laboratorio, los estudiantes han alcanzado una visión más amplia de las relaciones entre los sistemas operativos, la arquitectura y tecnología de computadores y la computación paralela y distribuida, aumentando la percepción de tales interacciones de un 83 % (encuesta previa) a un 92 %. Como resultado de aprendizaje, los estudiantes discernieron entre las tres alternativas de implementación de mutex e identificaron claramente los compromisos entre programabilidad, tiempo de ejecución y eficiencia.

IV. Conclusiones y Trabajo Futuro

Este artículo ha presentado un proyecto transversal cuyo objetivo es dotar al alumnado del Grado en Ingeniería Informática de una visión de conjunto de los sistemas informáticos, ya que la estructura actual del Grado en asignaturas aisladas tiende a romper las relaciones entre los distintos niveles de abstraccion.

Como caso de uso se ha presentado un sistema informático que implementa un trazador de rayos paralelo con cuatro niveles de abstracción introducidos en distintas

asignaturas del Grado. El desarrollo práctico de dicho sistema permite a los estudiantes experimentar con los conceptos de atomicidad, consistencia, paralelismo y concurrencia en los niveles de Aplicación, Biblioteca, Sistema Operativo y Arquitectura del Lenguaje Máquina. El proyecto se sustenta con la elaboración de los correspondientes enunciados y recursos de laboratorio para su implementación.

El proyecto propuesto se está llevando a cabo actualmente en las asignaturas elegidas del Grado. Por ahora, la retroalimentación recibida por parte del alumnado es alentadora. La mayoría de estudiantes coinciden en la necesidad de consolidar una visión de conjunto de los conceptos principales del Grado. Al terminar la experiencia de aprendizaje propuesta, los estudiantes han mostrado una mejora en la percepción integrada de los conceptos tratados. Además, en la sesión de laboratorio evaluada, los estudiantes también mostraron la adquisición de los conocimientos abordados en la sesión.

Como trabajo futuro, planeamos involucrar más asignaturas para fortalecer el proyecto propuesto y obtener un análisis más detallado de los resultados de aprendizaje de los estudiantes con las sesiones de laboratorio restantes.

AGRADECIMIENTOS

Este trabajo ha sido financiado por la Universidad de Zaragoza mediante el proyecto PIIDUZ_18_246, por el Ministerio de Ciencia, Innovación y Universidades mediante el proyecto TIN2016-76635-C2-1-R (AEI/FEDER, UE) y por el Gobierno de Aragón (Grupo T58_17R) y FEDER 2014-2020 "Construyendo Europa desde Aragón".

REFERENCIAS

[1] ACM, "Computer Engineering Curricula 2016 Curriculum Guidelines for Undergraduate Degree Programs in Computer Engineering," 2016.

[2] J. Kramer, "Is Abstraction the Key to Computing?," *Communications of the ACM*, vol. 50, no. 4, pp. 36–42, 2007.

[3] D. Ginat and Y. Blau, "Multiple Levels of Abstraction in Algorithmic Problem Solving," in *Proceedings of the ACM SIGCSE Technical Symposium on Computer Science Education*, 2017, pp. 237–242.

[4] C. Ferner, B. Wilkinson, and B. Heath, "Toward Using Higher-Level Abstractions to Teach Parallel Computing," in *Proceedings of the IEEE International Parallel and Distributed Processing Symposium Workshops*, 2013, pp. 1291–1296.

[5] J. Cuenca and D. Giménez, "A Parallel Programming Course Based on an Execution Time-Energy Consumption Optimization Problem," in *Proceedings of the IEEE International Parallel and Distributed Processing Symposium Workshops*, 2016, pp. 996–1003.

[6] V. Kumar, *Introduction to Parallel Computing*, Addison-Wesley Longman Publishing Co., Inc., 2nd edition, 2002.

[7] D. J. Sorin, M. D. Hill, and D. A. Wood, *A Primer on Memory Consistency and Cache Coherence*, Morgan & Claypool Publishers, 1st edition, 2011.

[8] A. Valero, D. Suárez Gracia, R. Gran Tejero, L. M. Ramos, A. Navarro-Torres, A. Muñoz, J. Ezpeleta, J. L. Briz, A. C. Murillo, E. Montijano, J. Resano, M. Villarroya-Gaudó, J. Alastruey-Benedé, E. Torres, P. Álvarez, P. Ibañez, and V. Viñals, "Exposing Abstraction-Level Interactions with a Parallel Ray Tracer," in *Proceedings of the Workshop on Computer Architecture Education (In Press)*, 2019.

[9] A. Valero, D. Suárez Gracia, R. Gran, A. Muñoz, J. Ezpeleta, J. L. Briz, L. M. Ramos, A. C. Murillo, E. Montijano, J. Resano, M. Villarroya-Gaudó, and V. Viñals, "Atomicidad, Consistencia, Paralelismo y Concurrencia en un Trazador de Rayos elaborado a lo largo del Grado en Ingeniería Informática," in *Actas de las XXIX Jornadas de Paralelismo*, 2018, pp. 201–207.

[10] H. Franke, R. Russell, and M. Kirkwood, "Fuss, Futexes and Furwocks: Fast Userlevel Locking in Linux," in *Proceedings of the Ottawa Linux Symposium*, 2002, pp. 479–495.

[11] A. Williams, *C++ Concurrency in Action*, Manning Publications, 2012.

[12] G. R. Andrews, *Concurrent Programming. Principles and Practice*, The Benjamin/Cummings Publishing Company, Inc., 1st edition, 1991.

[13] U. Drepper, "Futexes Are Tricky," 2011, `http://people.redhat.com/drepper/futex.pdf`.

[14] U. Drepper and I. Molnar, "The Native POSIX Thread Library for Linux," 2005, `https://akkadia.org/drepper/nptl-design.pdf`.

[15] A. Silberschatz, P. B. Galvin, and G. Gagne, *Operating System Concepts*, Wiley Publishing, 9th edition, 2012.

[16] W. Stallings, *Operating Systems: Internals and Design Principles*, Prentice Hall Press, 6th edition, 2008.

[17] R. H. Arpaci-Dusseau and A. C. Arpaci-Dusseau, *Operating Systems: Three Easy Pieces*, Arpaci-Dusseau Books, LLC, 1st edition, 2018.

[18] ARM, "ARM DS-5 Development Studio Examples," 2018.

[19] N4L, "N4L Newtons4th Ltd PPA500/1500 KinetiQ. User Manual," 2018, `https://www.newtons4th.com/wp-content/uploads/2014/07/PPA5xx_15xx-User-Manual-v2-91.pdf`.

[20] E. Veach, *Robust Monte Carlo Methods for Light Transport Simulation*, Ph.D. thesis, Stanford University, 1998.

[21] M. Pharr, W. Jakob, and G. Humphreys, *Physically Based Rendering: From Theory to Implementation*, Morgan Kaufmann Publishers Inc., 3rd edition, 2017.

Aplicaciones de CHIP-8, una máquina virtual de finales de los 70, en los estudios actuales de Ingeniería Informática

N.C. Cruz[1], J.L. Redondo[2], J.D. Álvarez[3] y P.M. Ortigosa[4]

Resumen— **Los planes de estudio del Grado en Ingeniería Informática se apoyan en el uso de emuladores, simuladores y herramientas simplificadas con las que enseñar a los alumnos conceptos fundamentales. El software y el hardware comercial actual son generalmente demasiado complejos como para estudiarse directamente. En este contexto, algunas plataformas antiguas podrían ser útiles a los profesores para ilustrar conceptos a bajo nivel hardware y software. Este trabajo propone distribuir entre las asignaturas del Grado en Ingeniería Informática varios micro-proyectos basados en CHIP-8, una máquina virtual de finales de los años 70. Están diseñados para enseñar a los alumnos: i) programación en ensamblador, ii) la forma en la que se programan los emuladores y simuladores, iii) los fundamentos del software independiente de la plataforma, y iv) las ideas básicas del funcionamiento de los compiladores y ensambladores. Además, teniendo en cuenta el creciente interés en las videoconsolas antiguas y sus juegos, la propuesta también es válida tanto para motivar a los alumnos como para atraer a nuevos estudiantes. Los proyectos descritos en este trabajo se han diseñado como tareas complementarias para enriquecer las asignaturas en las que se inscriben. Aunque el plan de estudios considerado es el de la Universidad de Almería, las propuestas pueden adaptarse a cualquier otra simplemente identificando las asignaturas equivalentes.**

Palabras clave— **Enseñanza, Ingeniería Informática, CHIP-8, Emulador, Máquina virtual, Videojuegos antiguos.**

I. Introducción

LOS emuladores y simuladores son herramientas populares en el Grado en Ingeniería Informática (GII). Por ejemplo, en la Universidad de Almería, los estudiantes de GII tienen que trabajar con el simulador de redes Packet Tracer [1] en varias asignaturas [2]. También se espera que usen herramientas como VirtualBox [3] para emular diferentes máquinas. Para programación móvil, los kits estándar de desarrollo incluyen emuladores de sus plataformas también, como por ejemplo, el Android SDK [4]. Este tipo de software aumenta significativamente la flexibilidad y alcance de las clases. Los alumnos pueden trabajar por su cuenta y no dependen de infraestructura real en los laboratorios [3]. Además, estas herramientas permiten estudiar un gran espectro de situaciones [5] sin riesgo para equipos reales y en asignaturas de corta duración.

[1]Dpto. de Informática, Universidad de Almería, ceiA3, e-mail: ncalvocruz@ual.es.
[2]Dpto. de Informática, Universidad de Almería, ceiA3, e-mail: jlredondo@ual.es.
[3]Dpto. de Informática, Universidad de Almería, CIESOL, e-mail: jhervas@ual.es.
[4]Dpto. de Informática, Universidad de Almería, ceiA3, e-mail: ortigosa@ual.es.

Independientemende de las herramientas, la mayoría de los planes de estudio de GII tienen un enfoque transversal. Sus asignaturas están orientadas principalmente a enseñar los fundamentos de las distintas áreas de interés. Se espera que los alumnos desarrollen los temas que les resulten más atractivos ya en el ámbito laboral o en programas de estudio posteriores. De hecho, y a causa de su evolución constante, tanto el hardware como el software comercial se han hecho cada vez más complejos y difíciles de estudiar de forma integral. Por consiguiente, las pruebas de concepto y los sistemas sencillos tienen un gran interés en el ámbito educativo. El sistema operativo MINIX [6], que fue específicamente diseñado con fines pedagógicos, es un buen ejemplo de esta idea. De forma similar, pero en hardware, en [7] se ha diseñado una familia de procesadores RISC para educación. El micro-computador Raspberry Pi, a pesar de haberse diseñado inicialmente para introducir a los niños en las Ciencias de la Computación, ha demostrado ser también una buena plataforma educativa para el GII [8], [9], [10].

Desafortunadamente, la mayoría de estudiantes no tienen interés en detalles a bajo nivel de las Ciencias de la Computación [10]. Por ejemplo, aunque la mayoría disfrutan programar en lenguajes de alto nivel de abstracción como Python y Java, la acogida de ensamblador e incluso C es mucho peor. Sin embargo, a la mayoría de estudiantes les gustan los videojuegos incluyendo los clásicos, que fueron programados a bajo nivel. De hecho, los juegos antiguos y la emulación de videoconsolas descatalogadas es realmente popular [11], [10]. En este contexto, el presente trabajo plantea que CHIP-8 [12], una arquitectura virtual de finales de los años 70, es una buena plataforma para motivar a los alumnos y complementar su conocimiento. En primer lugar, cuenta con sólo 35 instrucciones enfocadas a la programación de videojuegos, lo que la hace atractiva a los estudiantes a pesar de ser un lenguaje pseudo-máquina. En segundo lugar, CHIP-8 es especialmente adecuado para iniciarse en el desarrollo de emuladores, lo que no se aborda generalmente en GII. En tercer lugar, CHIP-8 puede usarse para introducir los fundamentos del software interpretado e independiente de la plataforma. En cuarto y último lugar, puede servir también para mostrar algunos conceptos de la compilación y los archivos binarios. En este trabajo se diseñan varios micro-proyectos basados en CHIP-8 y orientados a complementar el contenido de cier-

tas asignaturas de GII en la Universidad de Almería. No obstante, pueden adaptarse a cualquier otro programa de estudios compatible. Se diseñan como actividades modulares de forma que pueden incluirse en cualquier asignatura sin alterar sus contenidos.

La estructura de este documento es la siguiente: la sección II describe el diseño de CHIP-8. La sintáxis de Octo [13], un ensamblador para dicha plataforma, también se incluye. La sección III contiene los detalles de todos los proyectos educativos propuestos. La sección IV plantea las conclusiones y el trabajo futuro.

II. Visión general de CHIP-8

CHIP-8 es un lenguaje de programación hexadecimal e interpretado que fue creado por J. Weisbecker en 1978 y descrito en [12]. Pretendía ser una especie de lenguaje máquina de alto nivel que hiciera posible crear software, especialmente videojuegos, sin tener que programar realmente en lenguaje máquina específico. Se esperaba que cada una de sus instrucciones pseudo-máquina fuera ligada al código máquina de la plataforma por el intérprete de la misma.

CHIP-8 se diseñó inicialmente para el COSMAC VIP, pero fue posteriormente llevado a otras máquinas como la Telmac 1800 y las COSMAC ELF por sus características [14]. También tiene diversos descendientes como SCHIP, que incluye algunas instrucciones nuevas y más potencia [13]. CHIP-8 recibe aún hoy en día el interés de aquellos interesados en máquinas antiguas y entre los que quieren iniciarse en la emulación de arquitecturas [13], [14].

Como se destaca en [12], una vez existe un intérprete CHIP-8 para una cierta plataforma hardware, el desarrollo de software para la misma es significativamente más sencillo. Además, en contraposición a otros lenguajes, la estructura rígida de CHIP-8 permite intérpretes muy simples y, por consiguiente, su uso en máquinas de muy bajas prestaciones. En [12] se dan diversas métricas para apoyar esta afirmación. En primer lugar, un juego de tres en raya escrito para CHIP-8 requiere hasta 6 veces menos bytes de código que su versión equivalente en BASIC. En segundo lugar, el intérprete de CHIP-8 es 8 veces más pequeño que uno de BASIC. En tercer y último lugar, el intérprete de CHIP-8 se ejecuta en un sistema con 1.5 K de memoria y sólo un teclado hexadecimal mientras que el de BASIC necesita 8K y un teclado ASCII y mostrado alfanumérico.

La especificación de CHIP-8 como lenguaje está ligada a una máquina virtual homónima [13] que proporciona implícitamente el intérprete. De hecho, como se dice en [12], el desarrollo para CHIP-8 puede enfocarse simplemente como microprogramación haciendo uso de sus instrucciones pseudo-máquina. Esta máquina virtual cuenta con un único procesador que está directamente conectado a su memoria principal, dos temporizadores, un zumbador, un teclado hexadecimal y una pantalla monocromática. El lector interesado debería ser capaz de escribir su propio intérprete de CHIP-8 así como software para dicha plataforma a partir de la documentación disponible en [13], [14], [12]. No obstante, a continuación se incluye un resumen para hacer este trabajo autocontenido en la medida de lo posible.

El procesador de CHIP-8 cuenta con 16 registros de propósito generald e 8 bits etiquetados como V0, V1, ...VF (éste último se usa también por el sistema para indicar ciertas condiciones). Contiene un registro especial de 16 bits, I, usado para almacenar direcciones de memoria, y 2 de 8 bits, VS y VT, ligados a los temporizadores de sonido y retardo, respectivamente. Cuenta también con dos registros especiales gestionados por el sistema, uno de 16 bits, PC, que se usa como contador de programa, y otro de 8 bits, SP, que se usa como puntero de pila. Se pueden guardar hasta 16 direcciones de 16 bits en la pila para poder volver de llamadas a subrutinas, aunque realmente el número de niveles puede variar según la implementación [15].

La cantidad máxima de memoria que se puede soportar en CHIP-8 es 4096 bytes. Se direcciona por byte. Por tanto, cualquier dirección válida sólo requiere 12 bits. Sin embargo, la implementación inicial de CHIP-8 mapeaba directamente su memoria sobre la del hardware subyacente y en la que el propio intérprete, de 512 bytes, estaba también cargado. Por este motivo, CHIP-8 asume generalmente que el código de programa empieza en la dirección 0x200 (la 512 en decimal). Esta particularidad la heredan las implementacionesmodernas para mantener la compatibilidad con software antiguo [14], [15]. Ese espacio se usa actualmente para almacenar el conjunto de fuentes que la especificación del lenguaje define.

Los temporizadores de sonido y retraso funcionan a 60 Hz y decrementan los registros VS y VT, respectivamente, mientras el valor que contienen es mayor que 0. Se espera que el zumbador suene mientras que el valor de VS sea mayor que 0.

El teclado hexadecimal cuenta con 16 teclas etiquetadas desde 0 a F y serviría en teoría tanto para escribir como para usar el software.

La pantalla, monocromática como se anticipó, tiene 64 píxeles de ancho y 32 píxeles de alto. Su estado es persistente, lo que requiere 256 bytes de memoria que quedan bajo la gestión del intérprete pero no son parte del mapa de memoria de la máquina virtual. Los gráficos se representan como sprites codificados en bytes. Tienen 8 píxeles de ancho y hasta 15 de alto. Al dibujar, se calcula la función XOR entre cada píxel del sprite y del estado actual de la pantalla en su punto correspondiente. Si alguno de los píxeles cambió de 1 (activo) a 0 (inactivo) al representar el sprite, VF se pone a 1 (y a 0 en caso contrario). Toda máquina CHIP-8 debe proporcionar al software los sprites correspondientes a los símbolos hexadecimales, de 0 a F. Tienen 8 píxeles de ancho (aunque sólo se usan 4) y 5 de alto. La figura 1 explica el sistema de coordenadas que se sigue en la representación gráfica y resume los aspectos mencionados usando uno de los símbolos hexadecimales por defecto como ejemplo.

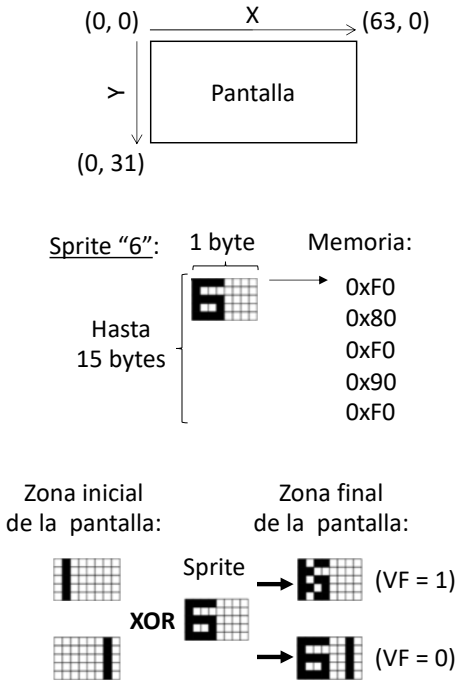


Fig. 1. Aspectos principales del dibujo en CHIP-8.

Como lenguaje, CHIP-8 define 35 instrucciones de 2 bytes para su procesador conceptual (aunque sólo 31 fueron documentadas en [12]). Como se pretendía en [12], es posible escribir software para CHIP-8 usando directamente su código máquina. Este es el motivo por el que CHIP-8 sólo requería un teclado hexadecimal para implementarse. De hecho, es simplemente una versión más cómoda de escribir conjuntos de unos y ceros. Sin embargo, recordar el significado conceptual de números y gestionar manualmente las direcciones de memoria son tareas costosas y propensas a errores. Por esta razón, es recomendable usar ensambladores [14] cuando sea posible. Este tipo de herramientas definen un lenguaje de tipo ensamblador para la arquitectura virtual de CHIP-8 y que asigna un mnemónico a cada código de operación (opcode). Gestionan además las direcciones de memoria automáticamente. CHIPPER [15] y Octo [13] son algunos ejemplos de ensambladores de código abierto para CHIP-8.

En este trabajo se recomienda el uso de Octo. Incluye un completo entorno de desarrollo (IDE) con emulador, desensamblador, depurador y funcionalidades para compartir los programas. Además, su documentación y tutoriales permiten aprender sin problemas a escribir código para CHIP-8. Como complemento, la tabla I lista todos los opcodes de CHIP-8 así como su mnemónico correspondiente en Octo. La nomenclatura usada se resume a continuación (nótese que los valores decimales deben convertirse a hexadecimal para incluirse en los opcodes):

- N es un entero en el rango [0, 15]
- NN es un entero en el rango [0, 255]
- NNN es una dirección de memoria en el rango [0, 4095]
- (V)X, Y se refiere a un índice de registro de propósito general en (0x)[0, F]

III. Actividades educativas basadas en CHIP-8

A pesar de su sencillez y antigüedad, CHIP-8 es una plataforma de desarrollo completa. Por tanto, aunque no debería reemplazar a contenidos más actualizados, CHIP-8 puede servir para motivar e introducir algunos conceptos en un contexto que es perfectamente comprensible y abarcable por los estudiantes.

Esta sección describe 5 actividades complementarias basadas en CHIP-8 para diferentes asignaturas y diseñadas con ese fin. Se pretende que sirvan como puente entre conceptos muy básicos, pero que podrían ser ignorados si no se tratan de forma explícita, y que mejoren la integración vertical del plan de estudios. Todas las asignaturas mencionadas pertenecen al plan de GII en la Universidad de Almería. Sin embargo, sus descripciones son lo bastante detalladas como para permitir identificar sus equivalencias con otros planes de estudio.

A. Proyecto 1: Bienvenido a un mundo binario

Este proyecto está diseñado para la asignatura Introducción a la Programmación, una materia obligatoria del primer cuatrimestre del primer curso (de un total de cuatro). Esta asignatura es de 6 créditos y 150 horas que se dividen en 45 de clase presencial y 105 de trabajo autónomo. Esta división horaria se mantiene también para el resto de asignaturas mencionadas en el artículo.

La asignatura pretende enseñar a los estudiantes los fundamentos de la programación. Abarca desde los conceptos generales de los algoritmos (variables, constantes, funciones y estructuras de control en pseudocódigo) hasta programación práctica usando Java bajo el paradigma de la orientación a objetos. Se trata probablemente de la asignatura del primer cuatrimestre más relacionada con lo que motivó a los estudiantes a elegir esta carrera.

En este contexto, la actividad propuesta está planificada para incluirse tras haber enseñado las condiciones y las operaciones de entrada salida, es decir, tras las primeras clases prácticas. Llegados a ese punto, los estudiantes deberían tener una definición teórica de programa como un conjunto de instrucciones que entienden los ordenadores. Deberían además haber desarrollado sus primeros programas. Por tanto, es un momento adecuado para darles una perspectiva más realista de que es el software en última instancia cuando sale del entorno de desarrollo y está listo para ser ejecutado.

El proyecto consiste en dar a los estudiantes un programa que simplemente escriba los bytes mostrados en la figura 2 a un archivo binario. Debería también permitir modificar los bits, al menos las zonas identificadas como "a" y "b". Se trata del código máquina de un programa CHIP-8 que imprime en pantalla "HELLO" o "BYE" dependiendo del valor de un cierto registro.

TABLA I
NEMONICOS DE OCTO (ADAPTADO DE [13]).

Opcode	Nemónico	Detalles
00E0	clear	Limpiar pantalla (todo a 0)
00EE	return	Volver de una subrutina
1NNN	jump NNN	Continuar desde NNN
2NNN	NNN	Llamar a la función en NNN
3XNN	if vX != NN then	Saltar instrucción sig. si VX = NN
4XNN	if vX == NN then	Saltar instrucción sig. si VX ≠ NN
5XY0	if vX != vY then	Saltar instrucción sig. si VX = VY
6XNN	vX := NN	VX = NN
7XNN	vX += NN	VX = VX + NN
8XY0	vX := vY	VX = VY
8XY1	vX |= vY	VX = VX OR VY (bit a bit)
8XY2	vX &= vY	VX = VX AND VY (bit a bit)
8XY3	vX ^= vY	VX = VX XOR VY (bit a bit)
8XY4	vX += vY	VX = VX + VY VF = 1 si VX+VY>0xFF o 0 s/n
8XY5	vX −= vY	VX = VX - VY VF = 1 si VX ≥ VY o 0 s/n
8XY6	vX >>= vY	VF = Bit menos significativo de VY Desplazar VY uno a la derecha. VX = VY
8XY7	vX =− vY	VX = VY - VX VF = 1 si VY ≥ VX o 0 s/n
8XYE	vX <<= vY	VF = Bit más significativo de VY Desplazar VY uno a la izquierda. VX = VY
9XY0	if vX == vY then	Saltar instrucción sig. si VX ≠ VY
ANNN	i := NNN	I = NNN
BNNN	jump0 NNN	Continuar desde NNN + V0
CXNN	vX := random NN	VX = Entero aleatorio ∈ [0, 255] AND NN (bit a bit)
DXYN	sprite vX vY N	Dibujar el sprite al que apunta I N es el número de bytes (filas) VF = 1 si hay colisión, 0 s/n (Fig. **??**)
EX9E	if vX −key then	Saltar instrucción sig. si la tecla identificada en VX está presionada
EXA1	if vX key then	Saltar instrucción sig. si la tecla identificada en VX no está presionada
FX07	vX := delay	VX = VT
FX0A	vX := key	Esperar tecla y guardar su valor en VX
FX15	delay := vX	VT = VX
FX18	buzzer := vX	VS = VX
FX1E	i += vX	I = I + VX
FX29	i := hex vX	Apuntar I al carácter hex. en VX
FX33	bcd vX	Descomponer el número en VX en centenas, decenas y unidades (decimal). Guardar en memoria (dir. I a I+2)
FX55	save vX	Copiar de V0 a VX (inc.) a memoria. V0 va a la dir. I, V1 a I+1, ... I = I + X + 1
FX65	load vX	Guardar en V0 a VX (inc.) los valores de memoria desde la dir. I. I = I + X + 1

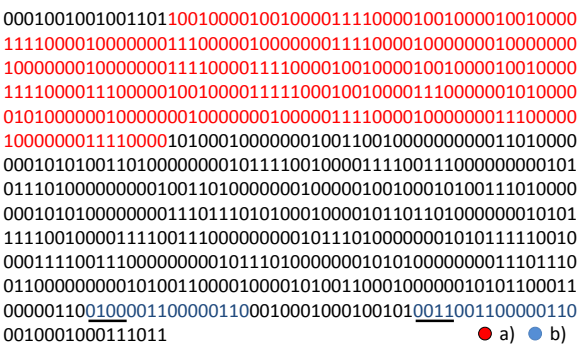


Fig. 2. Código CHIP-8 de muestra para el proyecto 1.

En primer lugar, el profesor hace que los alumnos creen y ejecuten el archivo binario en un emulador de CHIP-8 (Octo en nuestro caso). Después, el profesor recalca la idea de que los programas, en última instancia, no son más que conjuntos de instrucciones y datos codificados en cadenas binarias para una cierta plataforma. A continuación, invita a los estudiantes a analizar y modificar la zona “a” de forma creativa. Deberían darse cuenta de que contiene los mensajes que muestra el programa. Por ejemplo, los 40 primeros bits agrupados en filas consecutivas de 1 byte (véase la figura 1) representan la letra “H”. Tras eso, se les dice que intercambien los primeros cuatro bits de ambas partes de la región “b” y prueben el nuevo ejecutable. El cambio altera simplemente una secuencia de comparación (3XNN y 4XNN en la tabla 1), lo que hace que el programa muestre un mensaje alternativo. Finalmente, deberían pensar en las diferencias entre el código que escriben normalmente y el que necesitan las máquinas. Adicionalmente, si los alumnos muestran interés en la Seguridad Informática, el profesor podría invitarlos a discutir sobre el resultado y los riesgos potenciales de alterar los programas de esta forma (por ejemplo, el cracking).

Este proyecto enlaza lo que los alumnos escriben como código y lo que es realmente de una forma

práctica. Si no fuera por él, este aspecto quedaría fundamentalmente en un plano conceptual y no podrían apreciarlo directamente. Debería atraer su atención y mejorar su visión de la Programación. Ésto se logra de una forma simple y descriptiva gracias a la sencillez de CHIP-8 como computador. Puede llevarse a cabo en grupos durante una única sesión. El informe final de la actividad de cada grupo podría usarse como una nota extra de clase.

B. Proyecto 2: El enlace perdido entre los números y las letras

Este proyecto se ha diseñado para la asignatura Estructura y Tecnología de Computadores, una materia obligatoria del segundo y último cuatrimestre del primer curso. Su planificación se inicia con la enseñanza de Sistemas de Numeración, Álgebra de Boole, Lógica Combinacional y la circuitería relacionada (sumadores, multiplexores, codificadores, etc.). Después, incluye Lógica Secuencial y su circuitería (latches, flip-flops, etc.). Finalmente, se describe el sistema de memoria y la unidad central de procesamiento antes de hacer una introducción al lenguaje ensamblador de los procesadores ARM (MIPS en cursos anteriores) [16].

En este contexto, el propuesto proyecto está planificado para llevarse a cabo justo antes de empezar con la programación en ensamblador (una sesión antes, por ejemplo). En este punto, la mayoría de los alumnos deberían saber que los programas no terminan siendo más que combinaciones de números que codifican operaciones a realizar sobre datos. Los ensambladores proporcionan nombres simbólicos (mnemónicos) y gestionan aspectos como las direcciones de memoria. Este proyecto muestra la especificación de unos pocos opcodes (el formato numérico de las instrucciones) y su uso directo de una forma sencilla pero creativa. Después de realizarlo, los alumnos deberían ser capaces de apreciar las herramientas que proporcionan los ensambladores de una forma más práctica y con un nivel de comprensión más profundo.

La actividad empieza con la presentación a los alumnos de un procesador con sólo 8 opcodes y conectado a una memoria direccionable por bytes en la que se cargan los programas a partir de la dirección 0x200. Concretamente, el profesor explica simplemente CHIP-8 mencionando sólo sus 16 registros de propósito general de 8 bits, el registro especial I y sus opcodes 1NNN, 6XNN, 7XNN, ANNN, DXYN (incluyendo la representación gráfica, ver la figura 1), FX29, FX33 and FX65 de la tabla I. Después de dicha explicación, el profesor explica cómo escribir un programa para esa máquina directamente, con los opcodes. El programa simplemente dibujará un número y una letra.

La explicación debería basarse en la figura 3. Tal y como muestra, el programa empieza por saltar su propia sección de datos. Hace falta ajustar la parte "NNN" del opcode 1NNN para que lleve a la próxima instrucción válida (la que hace que V0 sea igual a 0

Dire.	Byte #i	Byte #i+1	Descripción	Octo
0x200	0x12	0x0A	Saltar a 0x20A	jump init
0x202	0x00	0x00	Reservar espacio y dibujar una 'N'	: room 0x0 0x0
0x204	0x00	0x88		0x0 : letter 0x88
0x206	0xC8	0xA8		0xC8 0xA8
0x208	0x98	0x88		0x98 0x88
0x20A	0x60	0x00	V0 = 0	: init v0 := 0
0x20C	0x61	0x00	V1 = 0	v1 := 0
0x20E	0x62	0x00	V2 = 0	v2 := 0
0x210	0x63	0x06	V3 = 6	v3 := 6
0x212	0x64	0x0A	V4 = 10	v4 := 10
0x214	0x65	0x6A	V5 = 106	v5 := 106
0x216	0xA2	0x02	Apuntar I a 0x202	i := room
0x218	0xF5	0x33	BCD de V5 (guardar desde I)	bcd v5
0x21A	0xF2	0x65	V[0,1,2] = Mem[I,I+1,I+2]	load v2
0x21C	0xF0	0x29	Apuntar I al símbolo de V0	i := hex v0
0x21E	0xD3	0x45	Pintar desde I en (V3,V4)	sprite v3 v4 5
0x220	0x73	0x06	V3 = V3 + 6	v3 += 6
0x222	0xF1	0x29	Apuntar I al símbolo de V1	i := hex v1
0x224	0xD3	0x45	Pintar desde I en (V3,V4)	sprite v3 v4 5
0x226	0x73	0x06	V3 = V3 + 6	v3 += 6
0x228	0xF2	0x29	Apuntar I al símbolo de V2	i := hex v2
0x22A	0xD3	0x45	Dibujar desde I en (V3,V4)	sprite v3 v4 5
0x22C	0x73	0x06	V3 = V3 + 6	v3 += 6
0x22E	0xA2	0x05	Apuntar I a 0x202 (la 'N')	i := letter
0x230	0xD3	0x45	Dibujar desde I en (V3,V4)	sprite v3 v4 5

Etiquetas de memoria

Fig. 3. Código de muestra para el proyecto 2.

en las direcciones 0x20B and 0x20C). De hecho, sólo hace falta modificar las partes variables de cada opcode para usarlos. Ésto implica gestionar manualmente las direcciones de memoria. En relación al bloque de datos saltado, contiene un conjunto de 5 bytes forman una letra "N" al dibujarse, y espacio para un array de tres posiciones para usarse por el opcode FX33 (BCD).

Después de explicar el procedimiento, se le pide a los alumnos que escojan un nuevo número y letra para que adapten el código manualmente en consecuencia. Se les puede dar un programa que vuelque los números de las instrucciones que escriban a un archivo binario, o bien usar directamente las herramientas binarias de Octo. Es interesante que se den cuenta de los problemas potenciales que hay al escribir código como lo están haciendo, mediante opcodes directamente. Se destacan por ejemplo la alta probabilidad de cometer errores y la dificultad para hacer cambios.

Esta actividad termina con el profesor mostrando a los alumnos el equivalente en ensamblador Octo al programa descrito, y que se incluye en la última columna de la derecha de la figura 3. Deberían apreciar cómo hay una correspondencia directa entre cada opcode de CHIP-8 y un mnemónico de Octo, además de que con el último no hay que gestionar las direcciones de memoria manualmente.

Esta actividad debería haber enriquecido la comprensión práctica de los alumnos acerca de lo que son el ensamblador y los ensambladores justo antes de empezar con ARM en la próxima sesión.

C. Proyecto 3: Pocos bytes pero mucha diversión

Este proyecto se ha diseñado para la asignatura Arquitectura de Computadores, una materia obligatoria del primer cuatrimestre del segundo curso. Extiende los contenidos de la asignatura Estructura y Tecnología de Computadores, considerada en la actividad anterior, estudiando los componentes de los computadores desde un mayor nivel de abstracción. Más concretamente, empieza por la definición de los

conceptos básicos de la Arquitectura de Computadores, rendimiento y las principales métricas consideradas. Después, aborda las estrategias fundamentales que se aplican para mejorar el rendimiento de los procesadores modernos incluyendo los caminos de datos segmentados y superescalares. A continuación, se estudia en detalle programación de procesadores ARM. La parte final de la asignatura explica la gestión de memoria, su jerarquía y el subsistema de entrada/salida.

En este contexto, como en el anterior, el proyecto propuesto está planificado para desarrollarse antes de comenzar con el ensamblador ARM. Pretende añadir a la planificación de la asignatura una actividad innovadora y amena que motive a los alumnos a programar en ensamblador. De hecho, suelen recordar pocos detalles de ese tipo de programación de la introducción a ARM hecha en Estructura y Tecnología de Computadores. Su experiencia tampoco ha sido normalmente muy positiva. Por tanto, el proyecto propuesto es bueno para preparar a los alumnos para volver a trabajar en ensamblador de una forma práctica, entretenida y sin alterar los contenidos de la asignatura. También incluye actividades opcionales para ganar puntos extra y que sirven para visualizar de forma práctica aspectos de la memoria y la eficiencia que se estudiarán más adelante en la asignatura.

El proyecto se ha diseñado como una única sesión guiada o tutorial. El profesor comienza mostrando a los alumnos una foto de un juego reciente y popular. A continuación, invita a los alumnos a imaginar que hubieran tenido esa idea en 1980. Seguidamente, les dice que van a implementarla para una plataforma de la época, concretamente CHIP-8, lo que se conoce como un "*demake*" en el ámbito de los videojuegos [17]. Entonces, el profesor les describe CHIP-8 y Octo como el ensamblador que van a usar para crear el juego. Si los proyectos previos se incluyeron en el plan de estudios de GII, los alumnos deberían estar ya familizarizados con algunos aspectos de CHIP-8 y Octo (por ejemplo, algunos opcodes y el emulador online). Sin embargo, este es el proyecto designado para incluir una introducción completa a la plataforma.

Es necesario proporcionar a los alumnos un material de referencia atractivo y completo pero también conciso. La explicación podría basarse en los contenidos incluidos en la sección II y no debería durar más del 25% de la sesión (hasta 30 minutos de una sesión estándar de 2 horas). Después de la explicación general, el profesor pasaría ya a describir, paso a paso, el código de un juego de CHIP-8 que imitara a uno reciente. Los estudiantes deben seguir la explicación desde sus puestos de trabajo. Para concluir la sesión, el profesor propone varias modificaciones opcionales al código proporcionado. Llevar a cabo dichas modificaciones debe hacerlos analizar el programa y servir para visualizar conceptos más complejos que se estudiarán posteriormente. Se les da aproximadamente 3 semanas para desarrollarlas y poder ganar los puntos extra.

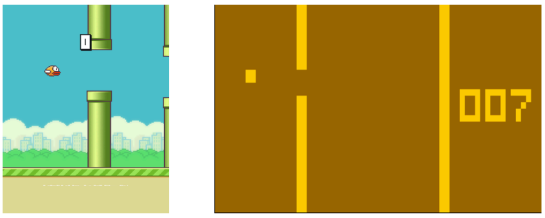


Fig. 4. Captura de pantalla de Flappy Bird (izquierda) y su clon programado para CHIP-8 (derecha).

```
: main
        loop
                showSplash
                v0 := 8
                waitV0
                v3 := 5
                if v3 key then game
        again
```

Fig. 5. Código del bucle principal del juego programado para el proyecto 3 [19].

De acuerdo con lo expuesto, el profesor debe seleccionar un juego moderno y obtener su equivalente en CHIP-8 para usarlo. El objetivo no debería ser demasiado ambicioso porque se pretende motivar a los alumnos y animar el programa de la asignatura, no añadir una carga adicional. De hecho, se debería poder explicar el código en una única sesión.

Hay muchos juegos que cumplen los requisitos previos entre los que están disponibles en el propio Octo. Sin embargo, los autores de este trabajo han desarrollado el suyo propio. Se llama Flappy Pong, y es un clon del popular juego para móviles Flappy Bird [18]. El código desarrollado está disponible públicamente en [19], y el lector interesado puede usarlo para llevar a cabo este proyecto. La figura 4 muestra una captura de pantalla del juego original (izquierda) y del clon ejecutándose en Octo (derecha). La figura 5 incluye un fragmento del código correspondiente al bucle principal del juego.

La explicación del juego motiva a los alumnos a aprender programación a bajo nivel y a dominar la arquitectura. Por ejemplo, si programamos el juego en ensamblador Octo, se pueden ahorrar 2 bytes sin finalizar la función *main* (ver la figura 5) con una instrucción *return* teniendo en cuenta que el programa se ejecuta en bucle infinito. De hecho, el binario final sólo ocupa 417 bytes mientras que el juego moderno real tiene varios megabytes. Teniendo en cuenta que la jugabilidad se mantiene intacta, ese detalle resulta muy sorprendente a los alumnos.

Se pueden también mencionar ciertas ideas sobre compiladores y ensambladores, no sólo sobre la gestión automática de las direcciones de memoria, sino también sobre su capacidad de procesado de código. Por ejemplo, en la figura 5 se puede ver una estructura *loop-again*. Sin embargo, esa construcción no aparece en la tabla I. La razón es que Octo proporciona algunas instrucciones virtuales sobre CHIP-8 que se convierten en compilación. Por ejemplo, el mencionado bloque *loop-again* se convierte en última

instancia en un simple salto al principio del bucle. Eso se puede apreciar fácilmente comparando la salida escrita en el binario con el equivalente al salto manual en la zona correspondiente.

Volviendo al código de muestra propuesto, se pueden proponer tres tareas. La primera es cambiar el número de puntos que hace que el juego acabe. De esta forma se fuerza a que los alumnos entiendan un programa dado en ensamblador. La segunda tarea es añadir un nuevo tipo de obstáculo que superar. Ésta requiere un conocimiento más profundo del código y obliga a los alumnos a entender y trabajar con las diversas instrucciones de CHIP-8. La tercera y última tarea, que es la más difícil, los reta a guardar los datos del juego (por ejemplo, el título mostrado al iniciar el programa) de forma más eficiente en memoria. Para hacerlo, necesitan reordenar y agrupar partes de los bytes que componen los datos permitiendo un código más simple y eficiente en última instancia. Esta estrategia también puede aplicarse para optimizar la pantalla final del juego, que muestra una copa de victoria sólo cuando se alcanza el máximo de puntos. Aunque esta idea del almacenamiento y recorrido eficiente de memoria se estudia más adelante en la asignatura trabajando con matrices, realizar esta tarea va a familiarizarlos con este tema de forma muy visual, amena y creativa.

Teniendo en cuenta además que los alumnos pueden descargar y usar luego el juego, les es posible también apreciar y pensar en las posibilidades que brindan los entornos de ejecución y las máquinas virtuales. De hecho, podrían poder jugar al juego desde sus propios ordenadores y teléfonos inteligentes simplemente usando un emulador de CHIP-8. Para que ésto sea posible, ese tipo de programas implementa el entorno definido por la especificación de CHIP-8. Esta idea es más difícil de asimilar con entornos más complejos como una máquina virtual de Java (virtualización) o un intérprete de Python (lenguaje interpretado).

Finalmente, como resultado complementario de esta actividad, los alumnos van a aprender los detalles de una máquina de videojuegos real y antigua. Ésto es positivo para su cultura específica.

Este proyecto ha sido ya incluido en el programa de la asignatura mencionada en la Universidad de Almería (curso 2018/19). El grado de satisfacción entre los estudiantes fue aproximadamente del 90%, y más del 60% trabajó en las tareas opcionales.

D. Proyecto 4: El tiempo vuela

Este proyecto se ha diseñado para la asignatura Sistemas en Tiempo Real, una materia optativa del segundo cuatrimestre del tercer curso. Esta asignatura, como se deduce de su nombre, trata sobre el ámbito de aplicación, el diseño y la implementación de sistemas en tiempo real. A grandes rasgos, ese tipo de sistemas son aquellos que tienen que ejecutar su función en intervalos precisos de tiempo, o en ciertos instantes, con unos recursos de cómputo limitados. Aspectos como la concurrencia y la tolerancia a fallos son críticos en los sistemas en tiempo real.

La asignatura se inicia con una introducción a los sistemas en tiempo real y a las técnicas de programación con las que se construyen. Después, se habla de las principales estrategias que existen para diseñarlos e implementarlos incluyendo planificación de tareas. Finalmente, se tratan y discuten varios casos prácticos.

En este contexto, el proyecto propuesto está planificado para incluirse en la parte inicial de la asignatura, cuando se describen los problemas prácticos de los sistemas de propósito general para cumplir los requisitos de las aplicaciones en tiempo real. Los alumnos deberían estar también trabajando en estimaciones teóricas del tiempo de ejecución en esa fase. La actividad pretende motivar a los alumnos a la vez que permitirles ver, de forma más práctica, las limitaciones de no usar sistemas en tiempo real con ese objetivo, y ajustar estimaciones.

El proyecto consiste en estimar el retraso potencial de un cronómetro simple programado para CHIP-8. En la figura 6 se incluye el código de una implementación que mide el tiempo con el registro de retardo de 60 Hz.

En CHIP-8, cada instrucción tarda un solo ciclo. Aunque la frecuencia de su procesador no se fijó en la especificación, a efectos prácticos estaba entre los 400 y los 1000 Hz según la máquina de la época. Sólo admite un flujo de ejecución, no soporta interrupciones y no permite definir prioridades. Por tanto, es un ejercicio simple para incluir en las relaciones iniciales de problemas teóricos. De hecho, a diferencia de los ejercicios puramente conceptuales, los alumnos podrían ver en este caso cómo funciona el programa y validar sus estimaciones. Para tal fin, deberían estar familiarizados con la plataforma de proyectos anteriores, y la mayoría de los emuladores como Octo admiten diferentes velocidades de ejecución.

E. Proyecto 5: Mi máquina, mis reglas

Este proyecto se ha diseñado para la asignatura Desarrollo Rápido de Aplicaciones, una materia del segundo cuatrimestre del cuarto curso. La asignatura trata el paradigma de desarrollo homónimo. Pretende también poner en práctica buenas prácticas de programación mediante el uso de patrones de desarrollo software y de herramientas y metodologías apropiadas.

La asignatura comienza con una introducción a las herramientas y metodologías de desarrollo ágil. Después, aborda el tema de los patrones de diseño centrándose en aquellos más convenientes para el desarrollo rápido de software (y usando Java). Posteriormente, se centra en servicios web y aplicaciones. El tema tratado después es el de herramientas específicas para desarrollo rápido prestando también a sistemas de virtualización y automatización. Tras eso, se explican varias herramientas y métodos de desarrollo de software basado en componentes. Finalmente, los estudiantes tienen que elaborar un proyecto propio para poner en práctica los conceptos

```
: separator 0x00 0x80 0x00 0x80
: digits 0 0 0

: subplot # Plot pair of numbers
        i := hex v1
        sprite va vb 5
        va += 5
        i := hex v2
        sprite va vb 5
        return

: plot
        va := 22
        i  := digits  # Where to write the BCD
        bcd v4              # Read MINUTES:
        load v2             # Load digits into v0-v2
        subplot
        va += 7
        i  := digits  # Where to write the BCD
        bcd v3              # Read SECONDS:
        load v2             # Load digits into v0-v2
        subplot
        return
```

```
: main
        v3 := 0 # Seconds
        v4 := 0 # Minutes
        vf := 0     # Buffer
        i := separator          # Plotting the SEPARATOR:
        va := 32
        vb := 13                # General height at plotting
        sprite va vb 4

        plot # Initial to be overriden
        : my_loop
                vf := 60
                delay := vf
                loop
                        vf := delay
                        if vf != 0 then
                again
                plot # Clear the previous one, updt and plot
                v3 += 1

                if v3 == 60 then v4 += 1
                if v3 == 60 then v3 := 0
                if v4 == 60 then v5 += 1
                if v4 == 60 then v4 := 0
                plot
        jump my_loop # Infinite loop
return
```

Fig. 6. Código de un cronómetro en CHIP-8 para el proyecto 4.

estudiados en la asignatura.

En este contexto, el profesor plantearía el desarrollo de un intérprete de CHIP-8 como uno de los proyectos finales disponibles para elegir. Como se puede ver en la sección II, aunque el sistema es simple, CHIP-8 define una arquitectura virtual completa con un procesador, una memoria, entrada/salida, gráficos y sonido. Hace falta también acceder al sistema de archivos para cargar los programas. Por tanto, hay margen para poner en práctica los patrones de diseño aprendidos mientras aprenden también los fundamentos de cómo funcionan una máquina virtual y un intérprete. En caso contrario, los estudiantes no llegarían a analizar realmente ese tipo de software y quedarían relegados a ser meros usuarios de los mismos.

Los autores de este trabajo han desarrollado su propia máquina virtual de CHIP-8, que puede encontrarse con su código en [20], y puede utilizarse para planificar la implantación de este proyecto.

Aunque no todos los alumnos se decidan por este proyecto en la asginatura, igualmente aprenderán las ideas principales de la virtualización y emulación durante la exposición final. Además, se espera también que este proyecto mejore la confianza como Ingenieros Informáticos de los estudiantes del último curso. Así pueden saber cómo crear el tipo de software que habrán usado previamente tantas veces, especialmente en su tiempo de ocio.

IV. Conclusiones y trabajo futuro

En este trabajo se han propuesto cinco proyectos educativos diseñados para distribuirse entre las distintas asignaturas del Grado en Ingeniería Informática. Pretenden reforzar la enseñanza de ciertas áreas y suprimir algunas lagunas conceptuales que no se tratarían explícitamente en caso contrario. Permiten visualizar lo que son los programas a efectos prácticos, las dificultades asociadas a la programación directamente a bajo nivel y los beneficios de las herramientas de desarrollo modernas. Igualmente, permiten observar las limitaciones de los sitemas de propósito general para funcionar en tiempo real y aprender cómo funcionan las máquinas virtuales y los intérpretes. Todos se basan en CHIP-8, una máquina virtual poco conocida de finales de los años 70. A pesar de ser sencillo, CHIP-8 es un computador completo, y su sencillez le da un gran potencial educativo. Además, la distribución de varios proyectos relacionados con el mismo entorno mejora la integración vertical del plan de estudios.

Como trabajo futuro, los proyectos que aún no se han implantado se van a incluir en las asignaturas para los que han sido concebidos. Además, como el plan de estudios actual del Grado en Ingeniería Informática en la Universidad de Almería no trata el diseño de lenguajes de programación y compiladores, se está elaborando un proyecto nuevo centrado en esa temática. Como los anteriores, se beneficia de la arquitectura “simple pero completa” de CHIP-8.

Agradecimientos

Este proyecto ha sido financiado por el Ministerio Español de Economía, Industria y Competitividad bajo el proyecto RTI2018-095993-B-100, y la Junta de Andalucía bajo el proyecto P12-TIC301. N.C. Cruz es beneficiario de una beca FPU concecida por el Ministerio Español de Educación (código

FPU14/01728).

Referencias

[1] J. Janitor, F. Jakab, and K. Kniewald, "Visual learning tools for teaching/learning computer networks: Cisco networking academy and packet tracer," in *6th International Conference on Networking and Services*. IEEE, 2010, pp. 351–355.

[2] J. F. R. Herrera, L. G. Casado, V. G. Ruiz, E. M. Garzón, J. F. S. Estrada, J. A. M. Naranjo, and P. M. Ortigosa, "Integration of cisco certification programs on computer networks subjects," *Experiencias Docentes en Redes de Computadores*, vol. 1, no. 1, 2015.

[3] P. Li, "Selecting and using virtualization solutions: our experiences with vmware and virtualbox," *Journal of Computing Sciences in Colleges*, vol. 25, no. 3, pp. 11–17, 2010.

[4] J. Steele and N. To, *The Android developer's cookbook: building applications with the Android SDK*, Pearson Education, 2010.

[5] J. Flochová, I. Zolotová, and D. Mudrončík, "Industrial tools and emulators in bachelor education," *IFAC Proceedings Volumes*, vol. 36, no. 10, pp. 231–236, 2003.

[6] A. S. Tanenbaum and A. S. Woodhull, *Operating systems: design and implementation*, vol. 68, Prentice Hall Englewood Cliffs, 1997.

[7] N. Bhardwaj, M. Senftleben, and K. Schneider, "Abacus: A processor family for education," in *Proceedings of the WESE'14: Workshop on Embedded and Cyber-Physical Systems Education*. ACM, 2014, p. 2.

[8] J. Kawash, A. Kuipers, L. Manzara, and R. Collier, "Undergraduate assembly language instruction sweetened with the raspberry pi," in *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*. ACM, 2016, pp. 498–503.

[9] S. J Matthews, J. C. Adams, R. A. Brown, and E. Shoop, "Portable parallel computing with the raspberry pi," in *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*. ACM, 2018, pp. 92–97.

[10] G. Ortega, E. M. Garzón, N. C. Cruz, J. L. Redondo, J. M. G. Salmerón, L. G. Casado, V. G. Ruiz, P. M. Ortigosa, C. Medina-López, J. J. Moreno, M. Ruiz-Ferrández, F. J. Orts, and S. P. Martín, "Educational strategies based on low-cost platforms in the area of Computer Engineering," in *Proceedings of ICERI2017 Conference*, 2017, pp. 7212–7218.

[11] S. Downing, "Retro gaming subculture and the social construction of a piracy ethic," *International Journal of Cyber Criminology*, vol. 5, no. 1, 2011.

[12] J. Weisbecker, "An easy programming system," *BYTE Magazine*, pp. 108–122, 1978.

[13] J. Earnest, "Octo, a high-level assembler for the CHIP-8 virtual machine," `https://github.com/JohnEarnest/Octo`, 2015.

[14] M. Mikolay, "Mastering CHIP-8," `http://mattmik.com/files/chip8/mastering/chip8.html`, 2015.

[15] D. Winter, "CHIP-8," `http://www.pong-story.com/chip8/chp8{\_}220.zip`, 1998.

[16] D. A. Patterson and J. L. Hennessy, *Computer Organization and Design ARM Edition: The Hardware Software Interface*, Morgan kaufmann, 2016.

[17] J. Vanderhoef, *Fans and Videogames: Histories, Fandom, Archives*, chapter NES Homebrew and the Margins of the Retro-gaming Industry, Taylor & Francis, 2017.

[18] Yongjian Wang, *A framework for developing network based games using Unity and Ice*, Ph.D. thesis, Universidad Politécnica de Madrid, 2017.

[19] N. C. Cruz, "E-Chip-8, a simple CHIP-8 emulator written in Java," `https://github.com/cnelmortimer/EChip-8`, 2016.

[20] N. C. Cruz, "Flappy Pong, a clone of Flappy Bird for CHIP-8," `https://github.com/cnelmortimer/FlappyPongC8/blob/master/FlappyPong.c8`, 2017.

Simulación de un procesador ARM para la enseñanza de Estructura de Computadores

S. Puertas-Martín[1], J. J. Moreno[1], F. J. Orts[1], N. C. Cruz[1], J. L. Redondo[1], E. M. Garzón[1] y P. M. Ortigosa[1]

Resumen— **En este artículo se expone el cambio realizado durante el curso académico 2018-2019 en la asignatura de Estructura de Computadores con la inclusión del procesador ARM que tan buena aceptación tiene entre los estudiantes. En concreto se ha utilizado una versión simplificada del procesador, denominada *ARM-Simple*, para facilitar el aprendizaje a los alumnos de primer curso. Para la simulación del procesador, se ha utilizado la herramienta *Digital*, un simulador de circuitos que posee un conjunto muy interesante de características que facilitan el diseño, construcción, análisis y validación de los distintos componentes. Se han propuesto un conjunto de actividades que abordarán los alumnos durante el trascurso de la asignatura. Estas actividades se les presentan en iteraciones, de forma que construyen el procesador incrementalmente añadiendo en cada iteración un componente nuevo. Para terminar, en la última iteración se les proporciona un código en alto nivel que deben convertir a código ensamblador, posteriormente a código máquina y ejecutarlo en los procesadores que hayan construido.**

Palabras clave— **ARM, Docencia, Estructura de Computadores.**

I. Introducción

La tecnología de computadores es una de las áreas actuales en las que mayores cambios se están produciendo y que mayor repercusión tiene en la sociedad. Uno de sus principales condicionantes se encuentra en la arquitectura subyacente. Nuevos materiales, técnicas o incluso disposición de los elementos posibilitan traspasar barreras que hasta ese momento impedían su desarrollo.

En el Grado en Ingeniería Informática de la Universidad de Almería se encuentra la asignatura de Estructura y Tecnología de Computadores que es impartida en el segundo cuatrimestre del primer año. Esta asignatura representa la primera toma de contacto de los alumnos recién llegados a la universidad con este hardware subyacente. Smartphones, relojes inteligentes, tablets son ejemplos de dispositivos que se han aprovechado de estos cambios. Si se centra la atención en sus denominadores comunes, se puede observar que comparten una arquitectura común, la misma que ha dominado todos los dispositivos móviles y embebidos desde hace más de una década: la arquitectura RISC ARM [1]. Algunas de las características que han posibilitado este auge son la disminución de calor y consumo que permiten ser ideales para dispositivos que requieren de una batería para su funcionamiento. Por tanto, dispositivos móviles, tablets, videoconsolas portátiles, Raspberrys [2] y Arduinos [3] incorporan procesadores ARM.

En base a esto, este año se ha decidido modificar los contenidos de la asignatura de Estructura de Computadores [4, 5]. Hasta el curso pasado, estaba centrada en el procesador MIPS. El procesador MIPS era adecuado para explicar conceptos de una manera sencilla pero no llegaba a convertirse en una motivación para el alumnado pues no podían poner en práctica los conocimientos que adquirían. Por ello, este año se ha reemplazado el procesador MIPS por el de ARM al igual que se han hecho en otros Grados en Ingeniería Informática de otras universidades [6–9]. Además, en clases de tecnología de muchos institutos ya se incluyen dispositivos como Raspberry Pi y Arduinos para realizar proyectos por lo que la entrada en la universidad y más concretamente en la asignatura de Estructura de Computadores le supone aprender a un nivel mucho más bajo como funciona el *juguete* que han estado utilizando años anteriores.

Estructura de Computadores no es la única asignatura donde los estudiantes trabajarán con procesadores ARM, sino que es la primera oportunidad que se les ofrece. En años posteriores continúan con Arquitectura de Computadores y Multiprocesadores teniendo en el horizonte el Máster en Ingeniería Informática en la especialización de Internet de las Cosas.

Este trabajo se organiza de la siguiente manera. En la Sección II se explica cómo ha sido adaptado un procesador ARM a la Asignatura de Estructura de Computadores y en la Sección III se describe el simulador *Digital* que ha sido utilizado para la realización de las prácticas. En la Sección IV se muestran las actividades que han sido planteadas en este primer año. Por último, la Sección V termina con las conclusiones y las nuevas ideas para el próximo año de la asignatura.

II. Procesador ARM-Simple

En la asignatura los alumnos desarrollan un procesador inspirado en la arquitectura ARM de 32 bits, que se ha denominado *ARM-Simple*. Para simplificar la implementación, se utiliza un conjunto limitado (aunque versátil) de instrucciones:

- Instrucciones de memoria: `LDR` y `STR`.
- Instrucciones de procesamiento de datos: `AND`, `ORR`, `ADD`, `SUB` y `CMP`
- Saltos: `B`

Cabe destacar que todas las instrucciones ARM

[1]Departamento de Informática, Universidad de Almería, ceiA3, e-mail: savinspm@ual.es, juanjomoreno@ual.es, francisco.orts@ual.es, ncalvocruz@ual.es, jlredondo@ual.es, gmartin@ual.es y ortigosa@ual.es.

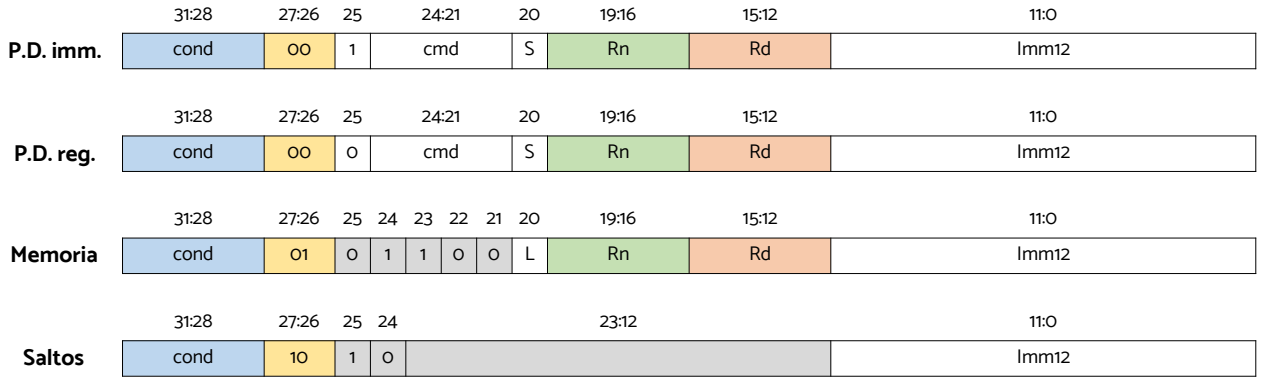


Fig. 1: Codificación de las instrucciones.

son condicionadas, lo que les proporciona una gran flexibilidad. La codificación de las instrucciones, mostrada en la Figura 1, también se simplifica para facilitar el diseño tanto del Camino de Datos como de la Unidad de Control. Por ejemplo, no implementamos rotaciones de inmediato en las instrucciones de procesamiento de datos, representando los mismos como valores de 12 bits en complemento a dos. Tampoco implementamos preindexado, postindexado o direccionamiento por bytes en las instrucciones de memoria.

A partir de este conjunto de instrucciones y por simplicidad se ha considerado implementar la versión monociclo del procesador de 32 bits que muestra la Figura 2. Se pueden observar dos partes: La Unidad de Control y el Camino de Datos. La Unidad de Control es un circuito combinacional que administra todas las señales de control del procesador. El Camino de Datos contiene todas las memorias y los elementos necesarios para obtener el siguiente estado a partir del actual. De izquierda a derecha, los elementos del Camino de Datos son los siguientes:

- Contador de Programa (PC): Aunque se suele añadir al banco de registros, en este caso se ha separado para clarificar su funcionamiento.
- Memoria de instrucciones (ROM): De solo lectura, con un único puerto de direcciones enlazado al puerto de lectura de las instrucciones.
- Banco de registros (*Register File*): Compuesto por 16 registros de 32 bits (incluyendo el registro especial R15 que contiene PC+8), dispone de dos puertos de lectura (RD1 y RD2) y un puerto de escritura (WD) controlados por A1, A2, y A3 respectivamente.
- Extensor de inmediato: Una de las simplificaciones que hemos hecho a nuestra microarquitectura es la reducción de los formatos del inmediato, por lo que el extensor únicamente tiene que tratar con inmediatos de 12 bits en CA2 que desplaza dos posiciones a la izquierda y convierte a 32 bits en CA2 (para los saltos) o que directamente convierte a 32 bits (para todas las demás operaciones).
- Unidad Aritmético-Lógica (ALU): Los alumnos implementan, partiendo de semisumadores de 1 bit, una ALU de 32 bits capaz de sumar, restar y realizar operaciones AND y OR. La ALU también genera cuatro *flags* (Negativo, Cero, Acarreo y Overflow), necesarios para la ejecución condicionada de instrucciones.
- Memoria de datos (*DataMemory*): Este circuito implementa internamente tres zonas de memoria. La primera zona es una memoria tradicional no volátil de lectura y escritura, equivalente a una EEPROM. La segunda zona es una memoria especial que muestra gráficamente (en representación RGB) lo que se escribe en ella. Esto permite implementar programas muy atractivos para el alumnado, como muestra la Figura 4. Finalmente también se conecta un terminal que muestra (en representación ASCII) lo que se escribe en su dirección de memoria.

III. Herramienta de simulación: *Digital*

En el proceso de renovación de la asignatura se consideró utilizar un nuevo simulador. Para ello se realizó un estudio preliminar sobre los distintos simuladores disponibles. En primer lugar, la búsqueda se centró en la selección de simuladores gratuitos y multiplataforma. El principal motivo de esta decisión fue no comprometer a los estudiantes a realizar sus prácticas en los equipos de laboratorio, sino que pudieran descargarlo e instalarlo en sus ordenadores personales. Los simuladores más interesantes que se encontraron en esta búsqueda fueron: *CircuitVerse* [10], *Logic.ly* [11], *CircuitLab* [12], *KTechLab* [13], *Logisim* [14] y *Digital* [15]. Después de analizar sus características, se redujo la selección a *Logisim* y *Digital* para finalmente decantarnos por este último.

Digital es un software de codigo libre para el diseño de circuitos digitales que se encuentra disponible en Github [15]. Está implementado en Java por lo que puede ser utilizado en cualquier sistema operativo con soporte para la JVM. A continuación, se muestran algunas de las mejoras que presenta *Digital* en comparación con *Logisim*:

- Conexión real del circuito: utiliza un enfoque basado en eventos. Esto significa que cada vez que se modifica la entrada de una puerta, se actualizan todas las entradas de las puertas y posteriormente se actualizan las salidas. Sin embargo, en determinados circuitos como puede ser

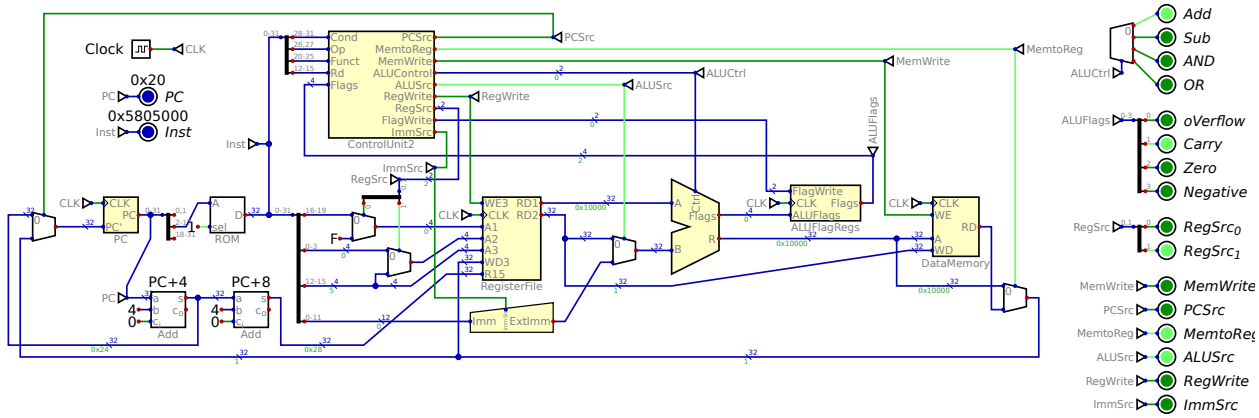


Fig. 2: El procesador ARM monociclo de 32 bits diseñado usando *Digital*.

un flip-flop RS está característica no es deseable ya que no se podría conseguir un estado estable del circuito. Para solucionar esto, incluye un proceso de simulación durante el proceso de inicialización inmediatamente después de encender el circuito. En este punto, solo se actualizan las entradas de la puerta modificada y posteriormente su salida. Para conseguir esto, incluye en el circuito una puerta con un *reset* especial de tal forma que al principio de la inicialización, el valor de esta puerta es bajo y una vez inicializado el circuito pasa a un valor alto. Esto implica que no se pueda modificar el circuito mientras está encendido, aunque lejos de ser un inconveniente, prepara a los estudiantes a trabajar de forma similar a como lo harían en un circuito real.

- Oscilaciones: Si bien en cada ciclo de reloj se ejecutan varias puertas, *Digital* tiene un modo de ejecución en el que se iluminan las puertas que están siendo alteradas en cada ciclo, por lo que permite seguir la propagación de cambios por todo el circuito y así detectar el componente que está provocando la oscilación.
- Depuración: proporciona un IDE ensamblador que puede usarse para controlar la simulación y cargar programas en ensamblador en el procesador simulado, iniciar el programa, etc. Además, permite iluminar la parte del circuito que se utiliza con cada instrucción por lo que es muy útil para depurar un programa en ensamblador en un procesador simulado.
- Síntesis de circuitos: Es capaz de generar circuitos combinatorios a partir de una tabla de verdad y viceversa. También se puede generar un circuito secuencial a partir de una tabla de transición de estados apropiada. Puede especificar tanto el circuito de transición como el circuito de salida. La minimización de las expresiones se hace por el método de Quine y McCluskey. La tabla de verdad también puede derivar de un circuito que contiene lógica combinatoria simple, flip-flops D o flip-flops JK, incluyendo la generación de la tabla de transición de estado. También es posible exportar un circuito a VHDL o Verilog para ejecutarlo en una FPGA.

Además de las características anteriormente mencionadas, hay otras dos muy interesantes. La primera de ellas está relacionada con las máquinas de estados de Mealy y Moore. *Digital* permite diseñar tu propia máquina de Moore y posteriormente generar las tablas de estados e implementarlas con distintos Flip-Flops los circuitos de manera automática. Esta característica es muy interesante para los estudiantes ya que aun teniendo que realizar manualmente el ejercicio completo, después puede comprobar la validez de sus soluciones antes de enviarlas a los profesores.

La segunda característica que se quiere destacar es la incorporación de test. Los estudiantes en el segundo cuatrimestre ya se encuentran familiarizados a realizar test de Java usando JUnit en las asignaturas de Introducción a la Programación (en el primer cuatrimestre) y Metodología de la Programación (paralela a la asignatura de Estructura de Computadores). Con *Digital* es posible realizar tests mediante las entradas y salidas de los circuitos. De esta forma, los alumnos pueden comprobar que su circuito sea correcto ejecutando el test una vez esté diseñado. En la Figura 3 se muestra un test realizado para la *ALU* explicada en la Sección II en el que se ha utilizado como parámetros de entrada 30000 datos distintos.

IV. Actividades

Las actividades planteadas a los estudiantes se encuentran estrechamente relacionadas con los contenidos impartidos en los grupos docentes. Hasta la tercera parte de la asignatura los alumnos han adquirido los conocimientos sobre circuitos lógicos básicos y están preparados para diseñar un procesador *ARM-Simple* completamente funcional similar al descrito en la Sección II.

La construcción del procesador se divide en 4 partes. En cada una de ellas se incorpora un nuevo elemento en base a las necesidades del código que sea necesario ejecutar. De esta manera los estudiantes entienden el motivo por el cual cada una de las partes (*ALU*, memoria, banco de registros, contador

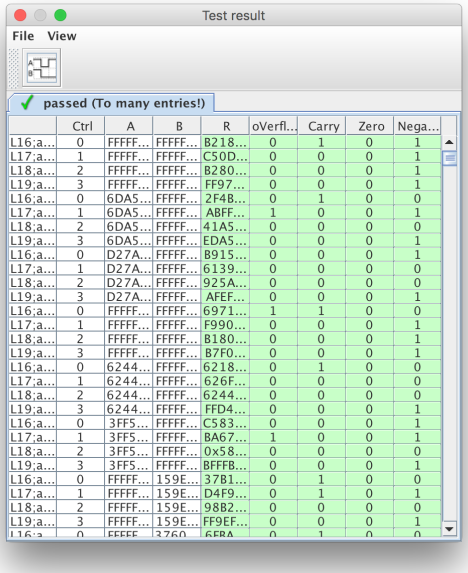


Fig. 3: Test de validación de la *ALU*.

de programa, etc) es importante y necesaria en un procesador.

El primer elemento que se le introduce es la *ALU*, que les permite hacer las operaciones aritmético-lógicas. Para construir el sumador/restador de 32 bits tienen que comenzar por un semisumador, posteriormente un sumador completo, y así hasta llegar al sumador de 32 bits mediante la concatenación de 32 sumadores completos. Finalmente tienen que modificar este último circuito usando puertas XOR para añadirle la funcionalidad de restador. Por otro lado, tiene que construir las operaciones AND y OR, también de 32 bits, e incluir estas 4 operaciones en un circuito con sus respectivas entradas, salidas y flags. Gracias a los circuitos embebidos de *Digital* este ensamblado resulta bastante sencillo y se obtiene una solución muy elegante gráficamente.

Una vez han construido la *ALU*, se les proporciona un código donde necesitan realizar cálculos con valores previamente calculados, por lo que tiene que almacenar dichos valores en algún componente. Aquí se les introduce el banco de registros. En el banco de registros pueden almacenar hasta 16 valores. En este punto se les pide que detallen cómo está construido el banco de registros, cuántos bits puede almacenar cada registro, cómo se encuentran relacionadas las entradas y las salidas, etc.

Llegados a este punto, le preguntamos qué sucede en los programas reales cuando es necesario tener almacenados más de 16 registros. Es aquí donde aparece el concepto de Memoria. Para trabajar con la memoria, se les propone ejercicios en los que primero deben leer el contenido de ésta, almacenar dichos valores en los registros, realizar algunas operaciones con estos datos que han obtenido y terminar almacenando nuevamente los datos en la memoria con las modificaciones realizadas.

Hasta este momento, todos los valores de entrada los han tenido que introducir manualmente. Obviamente, se les explica que en un procesador real las instrucciones deberán almacenarse en algún lugar y ser pasadas a otro componente que controle todos los elementos del procesador. Por tanto, aquí aparecen los conceptos de ROM (junto con el PC) y la Unidad de Control. Una vez tienen el procesador completo, se les facilitan dos archivos, uno con instrucciones de programa para la ROM y otro con datos precargados para la memoria. Gracias a una característica de *Digital*, los estudiantes cargan ambos archivos y si todo ha sido construido y conectado correctamente, debe de aparecer una ventana como la de la Figura 4 cuando ejecuten el circuito.

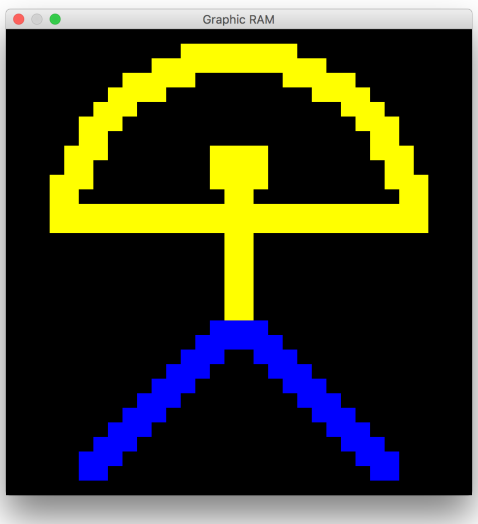


Fig. 4: Ejemplo de ejecución del programa de verificación de todos los componentes del procesador.

Por último, como tarea final y demostración de todos los conocimientos que han ido adquiriendo en las distintas actividades, así como en las clases teóricas, se les presenta a los alumnos un código en alto nivel similar al del Listado 1. En primer lugar, deben de convertir ese código a código ensamblador de ARM utilizando el conjunto de instrucciones reducido que han visto. Posteriormente deben pasar el código ensamblador a código máquina y por último a formato hexadecimal para poder ser insertado en la memoria ROM del procesador que han construido. Como las dos primeras instrucciones son de inicialización, saben que tienen que modificar también la memoria para cargar inicialmente esos valores cuando ejecuten de circuito.

V. Conclusiones

En este trabajo se ha expuesto el nuevo enfoque de la asignatura de Estructura de Computadores. Este cambio ha estado centrado en la sustitución del procesador MIPS por uno ARM que permitiese a los estudiantes tener un contacto más cercano del hardware que estudian. Este procesador *ARM-Simple* ha sido reducido en instrucciones para facilitar el aprendizaje a los estudiantes en esta primera toma de con-

```
int dividendo = data[0];
int divisor = data[1];
int conciente = 0;

while(dividendo >= divisor){
    dividendo = dividendo - divisor;
    cociente = cociente + 1;
}

data[2] = cociente;
data[3] = resto;
```

List. 1: Código inicial del ejercicio a convertir en código ensamblador y máquina.

tacto con procesadores reales. Una herramienta fundamental ha sido el simulador *Digital* que gracias a su interfaz actual y algunas de sus características ha sido recibido de forma positiva por los estudiantes y su forma de trabajar. Por último, se han propuesto un conjunto de actividades organizadas en iteraciones que, mediante la realización de cada una de ellas, permiten finalmente diseñar el procesador *ARM-Simple* completo y entender el funcionamiento y la utilidad de cada una de sus partes.

Para futuros cursos académicos se quieren incorporar las FPGA como último paso una vez se haya diseñado el procesador en el simulador. La idea detrás de esto resulta de la posibilidad que brinda *Digital* de exportar los circuitos al formato VHDL.

Agradecimientos

El presente trabajo ha sido financiado por el Ministerio de Economía y Competitividad de España mediante el proyecto RTI2018-095993-B-100. Savíns Puertas-Martín, Juan José Moreno y Nicolás C. Cruz son beneficiarios del programa español "Formación de profesorado Universitario", financiado por el Ministerio de Educación, Cultura y Deporte.

Referencias

[1] D. Seal, *ARM architecture reference manual*, Pearson Education, 2001.

[2] "Raspberry Pi Foundation," `https://www.raspberryi.org/`, Último acceso: 2019/06/10.

[3] "Arduino," `https://www.arduino.cc/`, Último acceso: 2019/06/10.

[4] D.A. Patterson and J.L. Hennessy, *Computer Organization and Design ARM Edition: The Hardware Software Interface*, Morgan Kaufmann Publishers, Inc., 2016.

[5] S. Harris and D. Harris, *Digital Design and Computer Architecture: ARM Edition*, Morgan Kaufmann Publishers Inc., 2015.

[6] R. Asenjo, S. González, F. Corbera, A. Navarro, A. Rodríguez, J. Villalba, and E. Hendrix, "Motivando al alumno de ingeniería mediante la plataforma Raspberry Pi," *Avances en arquitectura y tecnología de computadores. Actas de las Jornadas SARTECO*, pp. 313–320, 2017.

[7] C. Camarero, E.Z. Suarez, E. Stafford, F. Vallejo, and C. Martínez, "Enseñanza práctica de estructura y organización de computadores con Raspberry Pi," *Avances en arquitectura y tecnología de computadores. Actas de las Jornadas SARTECO*, pp. 305–312, 2017.

[8] A.J. Villena, R. Asenjo, and F. J Corbera, "Prácticas de ensamblador basadas en Raspberry Pi," *Arquitecturas Emergentes*, 2015.

[9] S. Barrachina, M. Castillo, G. Fabregat, J.C. Fernández, G León, J.V. Martí, R. Mayo, and R. Montoliu, *Introducción a la arquitectura de computadores con QtARMSim y Arduino*, 2018.

[10] "CircuitVerse," `https://circuitverse.org/`, Último acceso: 2019/06/10.

[11] "Logic.ly," `https://logic.ly/`, Último acceso: 2019/06/10.

[12] "CircuitLab: Circuit simulation and schematics.," `https://www.circuitlab.com/`, Último acceso: 2019/06/10.

[13] "KTechLab," `https://sourceforge.net/projects/ktechlab/`, Último acceso: 2019/06/10.

[14] "Logisim: A graphical tool for designing and simulating logic circuits," `http://www.cburch.com/logisim/`, Último acceso: 2019/06/10.

[15] "Digital: A digital logic designer and circuit simulator," `https://github.com/hneemann/Digital`, Último acceso: 2019/06/10.

Evaluación de prestaciones

Evaluación del módulo de estimación de movimiento basado en FPGA para el codificador de vídeo HEVC

Otoniel López-Granado, Roberto Gutiérrez, Estefanía Alcocer, Hector Migallón y Manuel P. Malumbres[1]

Resumen— **La estimación de movimiento es una de las tareas más costosas del codificador de vídeo HEVC debido principalmente a, a) un mayor número de modos de particionado del Coding Tree Unit, b) la presencia de múltiples frames de referencia y c) el tamaño variable de los Coding Units en comparación con su predecesor H264/AVC. Además, HEVC utiliza una estimación de movimiento de tamaño de bloque variable para aumentar la eficiencia en la codificación. En este trabajo se ha diseñado y evaluado sobre una plataforma System-on-Chip específica, un módulo de estimación de movimiento hardware, teniendo en cuenta tanto el rendimiento en compresión cuando se aplican diferentes tamaños de Coding Tree Units, como el impacto de las transferencias del DMA en el tiempo total de codificación.**

Los resultados muestran que el tiempo global de codificación se podría reducir un 77% usando nuestro módulo hardware, siendo este tipo de aceleradores, una solución de bajo coste interesante para acelerar los codificadores de vídeo de última generación.

Palabras clave— **Codificación de vídeo, HEVC, FPGA, Estimación de movimiento, Particionado asimétrico.**

I. Introducción

EL estándar de codificación de vídeo HEVC (High Efficiency vídeo Coding)[1] desarrollado por el Joint Collaborative Team on Video Coding (JCT-VC) en 2013, surgió para reemplazar a su predecesor H.264/AVC [2] con la intención de lidiar con las últimas tendencias del mercado multimedia como son los contenidos digitales de muy alta resolución (4K y 8K) así como con la mejora en la profundidad de color (10 bits). HEVC es capaz de comprimir el doble que su predecesor H.264/AVC para una calidad visual equivalente[3].

En lo referente a complejidad computacional, el codec HEVC es mucho más complejo que el codec H.264/AVC[4]. Al igual que en los anteriores estándares de codificación de vídeo, la Estimación de Movimiento (ME) es la parte más compleja computacionalmente hablando, requiriendo más del 90% del tiempo de codificación[5]. En HEVC, la ME es mucho más compleja, debido a las nuevas características incluidas en este estándar como son el uso de un mayor número de modos de particionado del Coding Tree Unit o la presencia de múltiples frames de referencia, en comparación con el anterior estándar de codificación de vídeo H264/AVC. Además, HEVC adopta un nuevo esquema de estimación de movimiento de tamaño de bloque variable (Variable Block Size Motion Estimation (VBSME)) para mejorar la eficiencia en la codificación, a expensas de un incremento en el coste computacional.

En la literatura podemos encontrar varias propuestas de arquitecturas hardware para acelerar el módulo de ME de HEVC. El bloque Integer-pel Motion Estimation (IME) es el encargado de la estimación de movimiento, por tanto, la mayoría de las propuestas del estado del arte se centran en el algoritmo de búsqueda de movimiento pues requiere la mayor parte del tiempo del bloque IME. Habitualmente el algoritmo de búsqueda de movimiento más usado en las implementaciones hardware es la búsqueda completa o Full Search (FS). Este algoritmo busca en todos los puntos del área establecida de un frame de referencia y en consecuencia, obtiene el resultado óptimo, es decir, obtiene el vector de movimiento que minimiza el error residual del CTU buscado.

Diversos autores como en [5], [6], [7], [8] y [9] han desarrollado bloques IME hardware usando el algoritmo FS. En [5], se propone una unidad para el calculo del SAD (Sum of Absolute Differences) implementada en una FPGA (Field-Programmable Gate Array) capaz de evaluar todos los posibles modos de particionado de un CTU excepto los modos asimétricos. Los autores establecen un área de búsqueda menor que la establecida en el estándar HEVC obteniendo una velocidad de codificación de 30 fps para vídeos de resolución 2k. En el módulo hardware presentado en [7], el tamaño máximo de CTU se reduce a 32x32 con un área de búsqueda de ± 23 pixels. Esta arquitectura alcanza los 30 fps para resoluciones de 1080p. En [8], se estudia el efecto de diferentes áreas de búsqueda, alcanzando un frame rate máximo de 57 fps para resoluciones de 720p.

En un trabajo previo [10], los autores presentaron una nueva arquitectura hardware que realiza el cómputo de IME usando tecnología FPGA. Esta nueva arquitectura se basa en dos técnicas innovadoras: la primera es una nueva estructura de árbol de sumadores para el calculo del SAD y la segunda es una nueva forma de recorrer la memoria. Esta propuesta es capaz de codificar 116 fps y 30 fps para resoluciones 2K y 4K respectivamente. El nuevo árbol de sumadores de SAD realiza las sumas en el primer nivel del árbol de particionado del CTU, comenzando con el mayor tamaño de CTU y dividiendo la cantidad de sumas a realizar en el siguiente nivel. Esta aproximación es muy diferente al resto de

[1]Dpto. de Ingeniería de Computadores, Univ. Miguel Hernández

propuestas encontradas en la literatura donde normalmente primero se divide el CTU en los tamaños más pequeños para ir acumulando sucesivamente, requiriendo un mayor número de pasos para obtener todos los SADs. Con esta nueva propuesta se maximiza el uso de los recursos de la FPGA obteniendo la mínima latencia posible a la hora de calcular los SADs de todas las particiones del CTU. De esta forma, los SADs que corresponden a las particiones asimétricas se obtienen de manera rápida y eficiente. Con respecto a la segunda técnica innovadora, la nueva forma de recorrer la memoria, se usan una serie de registros de desplazamiento reconfigurables y elementos de procesamiento para el almacenamiento de los pixels necesarios tanto de la referencia como del CTU actual, de manera que se mantienen siempre disponibles para calcular tanto los SADs como los vectores de movimiento (MV) de un CTU. Con esta estrategia, los autores evitan accesos a memorias externas ya que la información disponible se reutiliza reconfigurando los desplazamientos de manera eficiente.

En este trabajo, se ha implementado y evaluado el diseño del módulo IME presentado en [10] aplicado a una placa de evaluación específica. El diseño se ha variado ligeramente para obtener tres posibles implementaciones: a) una unidad que trabaja con el tamaño máximo de CTU de 32x32 pixels (CTU_32), b) cuatro unidades CTU_32 trabajando en paralelo y, c) una unidad que trabaja con el tamaño máximo de CTU a 64x64 pixels (CTU_64). Todos ellos usan el área de búsqueda propuesta por el estándar de vídeo HEVC. Hemos evaluado las tres versiones analizando los recursos hardware requeridos, la frecuencia de operación , el retardo total de los módulos hardware (transferencias DMA de entrada, cómputo de la estimación de movimiento y transferencias de DMA de salida), y el rendimiento en R/D (Rate/Distortion) de las propuestas hardware con respecto al software de referencia HEVC.

El resto del artículo se organiza de la siguiente manera. La Sección II muestra una breve descripción del diseño de la arquitectura propuesta mientras que en la Sección III, se presentan los resultados de los experimentos y se analizan los resultados de nuestro diseño hardware sobre la placa de evaluación. Finalmente, en la Sección IV se presentan las conclusiones y trabajo futuro.

II. Descripción de la Arquitectura Hardware

En esta sección presentamos una breve descripción del diseño IME sobre una plataforma System-On-Chip (SoC) que consiste en dos partes bien diferenciadas, un Processing System (PS) basado en un procesador ARM y algunos periféricos como Ethernet, USB, etc. y, un FPGA Programmable Logic (PL). Nuestra arquitectura ha sido modelada en VHDL y sintetizada, simulada, implementada y evaluada sobre la placa Xilinx SoC, Zynq-7 Mini-ITX Motherboard XC7Z100 (xc7z100ffg900-2). Para asegurar el correcto funcionamiento de nuestro diseño, éste se ha contrastado con el software de referencia del codificador HEVC HM 14[11].

En la arquitectura propuesta, el procesador ARM se encarga de las transferencias entre el módulo IME SAD y la memoria Double Data Rate (DDR) que almacena tanto la ventana de referencia como el CTU actual, a través de un módulo Direct Memory Access (DMA). El procesador ARM trabaja a 666.66 MHz, y la DDR a 533.33 MHz, mientras que la frecuencia de reloj del PL se ve restringido a la frecuencia máxima del módulo SAD HEVC responsable del calculo de IME.

En lo que concierne al proceso IME, cada frame del vídeo se subdivide en unidades básicas de codificación llamadas CTUs. La estructura de codificación de HEVC consta de CUs (Coding Units) de tamaño máximo 64x64 pixels, el mismo tamaño de los CTUs (Coding Tree Units), que pueden dividirse de forma recursiva en regiones cuadradas hasta alcanzar el tamaño de 8x8 pixels. Cada CU consiste en Prediction Units (PUs) cuyo tamaño puede variar desde el máximo tamaño del CU hasta 4x8 o 8x4 para la predicción Inter, pudiendo dividirse en 8 modos de particionado[12]. En nuestra propuesta, el módulo SAD HEVC encargado del calculo IME puede configurarse para trabajar con tamaños de CTU de 64x64 y 32x32. Para el caso de tamaño de CTU 64x64, el PL puede trabajar a 200 MHz mientras que con el tamaño de CTU 32x32 la frecuencia de reloj del PL queda restringida por la placa de evaluación a 250 MHz, aunque nuestro módulo podría trabajar a 333 MHz. Por lo tanto, la frecuencia máxima queda limitada a 200MHz para poder evaluar ambos tamaños de CTU.

Nuestro módulo SAD HEVC consiste en a) áreas de memoria interna para alojar los pixels del CU actual y los pixels correspondientes al área de búsqueda del frame de referencia, b) un bloque de distorsión donde los pixels de ambos CUs se restan, c) un bloque que corresponde al árbol de sumadores de SAD (Sum of Absolute Difference), y d) un bloque acumulador comparador que guarda el menor valor de SAD y su vector de movimiento (MV) correspondiente para todas las posibles particiones del CU, tal y como se muestra en la Figura 1. Para un mayor detalle del módulo véase [10].

En la Tabla I, se muestran los recursos hardware utilizados para implementar nuestro módulo SAD HEVC para tamaños máximo de CTU de 64x64 y 32x32, respectivamente, sobre una placa Zynq-7 Mini-ITX Motherboard XC7Z100 FPGA. Como se puede apreciar, nuestro módulo SAD HEVC necesita el 56.5% y el 15.3% del área total para las implementaciones CTU_64 y CTU_32, respectivamente. Además, la implementación 4xCTU_32 capaz de realizar el cómputo de 4 CTUs de 32x32 al mismo tiempo, utiliza un 59.7% del área disponible.

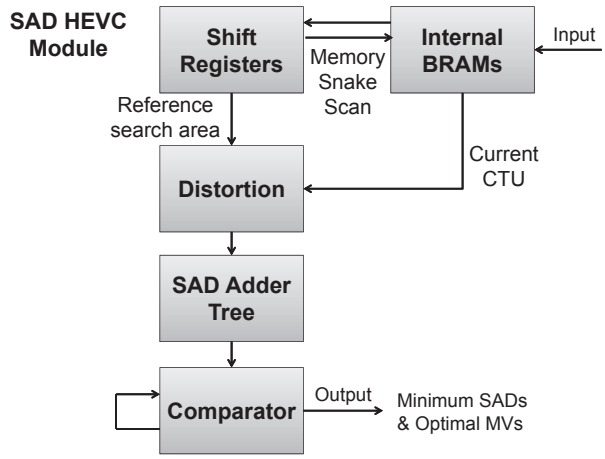


Fig. 1. Módulo hardware SAD HEVC

TABLA I
UTILIZACIÓN DE RECURSOS SOBRE MINI-ITX

	LUTs	Flip-flops	Block-RAMs
CTU_64	156880	180249	41,5
CTU_32	42405	50466	25,5
4xCTU_32	165700	190240	102
Disponible	**277400**	**554800**	**755**

III. EXPERIMENTOS

La estimación de movimiento es una tarea integrada en el módulo de predicción Inter del codificador HEVC. Para los experimentos se ha utilizado el modo de codificación Low Delay P (LD-P). En el modo LD-P, el primer frame se codifica como un cuadro Intra, mientras que el resto de frames se codifican como frames Inter unidireccionales. Este tipo de codificación es la más utilizada para sistemas de vídeo conferencia, y está diseñado para una comunicación en tiempo real. Usando el modo de codificación anterior, hemos realizado varios experimentos de nuestro módulo HEVC IME para ver cómo influye el tamaño del CTU en la eficiencia de la codificación (R/D), así como en el tiempo de codificación del codificador HEVC. Se han elegido los tamaños máximos de CTU de 64x64 y 32x32, y sus correspondientes áreas de búsqueda (SR) al 100% del tamaño del CTU. Se han utilizado cuatro secuencias de vídeo elegidas de entre las sugeridas en las condiciones comunes de test para HEVC[13]: RaceHorses (832x480-30 fps), ParkScene (1920x1080-24 fps), Traffic (2560x1600-30 fps) y PeopleOnStreet (2560x1600-30 fps). Para realizar estos test hemos usado el software de referencia HEVC HM 14 [11]. Este software de referencia se ha compilado utilizando Visual Studio 2015 y se ha ejecutado sobre una plataforma Intel Core i7-6800K CPU 3.40GHz con 16GB RAM.

TABLA II
COMPARACIÓN DE % BD-RATE ENTRE TAMAÑOS DE CTU DE 64X64 Y 32X32

Secuencia de vídeo	% BD-Rate
RaceHorses	2.4
ParkScene	3.8
PeopleOnStreet	3.7
Traffic	4.3

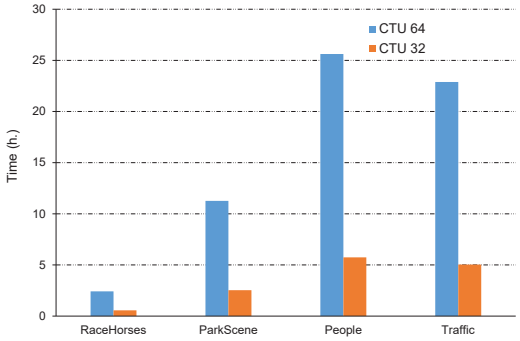


Fig. 2. Tiempo de codificación (30 frames) para CTU de 64x64 y 32x32 para todas las secuencias de vídeo.

Primeramente se ha analizado el impacto del tamaño máximo de CTU en términos de R/D usando la métrica Bjontegaard (BD-rate) [14]. Para obtener el valor de BD-rate se han comprimido todas las secuencias de vídeo a cuatro niveles de compresión (QP): 22, 27, 32, y 37. La curva R/D de referencia es la obtenida con el software de referencia usando un CTU de tamaño 64x64. Como podemos ver en la Tabla II se obtiene mejor calidad cuando usamos el CTU de tamaño 64x64. El uso de un CTU de menor tamaño tiene una penalización máxima de un 4.3% en términos de BD-rate, siendo esta penalización mayor en las secuencias de vídeo de alta resolución.

Después de la evaluación del R/D, evaluamos el impacto del tamaño de CTU en el tiempo de codificación. En la Figura 2 se muestra el tiempo total de codificación para comprimir 30 frames de todas las secuencias de vídeo evaluadas usando tamaños de CTU de 64x64 y 32x32 y para a un QP 37. Como se puede ver, el tiempo de codificación para el tamaño máximo de CTU de 64x64 es más de 4 veces mayor que el necesario para el caso de CTU de tamaño 32x32, necesitando más de 22 horas para comprimir una secuencia de resolución 4k. También, en la Figura 3, se muestra el porcentaje de tiempo total requerido por el algoritmo Full Search para realizar ME deshabilitando la optimización del R/D. Dependiendo del tamaño máximo de CTU y del parámetro de cuantización (QP), el porcentaje de tiempo utilizado por el proceso de ME va desde el 85% hasta el 97.8% en nuestros test. Estos resultados concuerdan con los obtenidos en [5]. Como era de esperar, el codificador requiere más tiempo en el módulo IME cuando el tamaño de CTU es mayor. Por lo tanto, tiene sentido la implementación de un diseño hardware para el cómputo del IME con el fin de reducir el tiempo total de codificación lo máximo posible.

Con el fin de evaluar el impacto de la inclusión de nuestro IME hardware en el HEVC, hemos medido el número de ciclos requeridos para a) la transferencia

TABLA III
CICLOS NECESARIOS Y TIEMPO DEL MÓDULO HARDWARE IME

	DMA Envío	DMA Recepción	SADs y MVs Cómputo	Ciclos totales	Tiempo Total (ms) a 200MHz
CTU_32	10069	151	4139	14359	0.07179
CTU_64	44276	605	16400	61281	0.30640

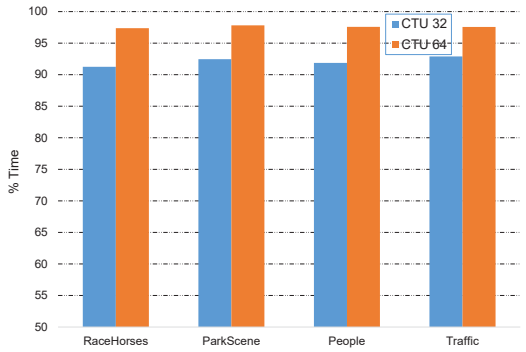


Fig. 3. Porcentaje del tiempo total del SW de referencia requerido por el módulo SAD en el algoritmo Full-Search.

tanto del CTU como de la ventana de referencia, b) el procesamiento del IME y c) la transferencia de los SADs y vectores de movimiento de todos los PUs. En la Tabla III se puede ver el número de ciclos requeridos para la realización del proceso IME para un determinado CTU, donde, como se ha comentado previamente, todos los SADs y MVs de todos los PUs incluidas las particiones asimétricas se calculan a la vez. De esta manera, el tiempo requerido para procesar un CTU será el tiempo para transferir el CTU y la ventana de búsqueda, el tiempo de procesamiento del módulo HEVC SAD, y el tiempo necesario para devolver los resultados. Como podemos ver, la mayor parte del tiempo necesario para un CTU está en las transferencias de DMA tanto del CTU como de la ventana de búsqueda desde la memoria DDR a la memoria interna del módulo hardware IME. En la Tabla III vemos el tiempo total requerido por nuestro módulo hardware IME para realizar la estimación de movimiento de un CTU. Considerando estos tiempos, el módulo hardware IME es capaz de procesar 13959 CTUs por segundo para el tamaño de 32x32 y 3264 para el tamaño de 64x64 a 200MHz.

TABLA IV
FRAMES DE REFERENCIA USADOS PARA UN GOP

Frame 1	-1 -5 -9 -13
Frame 2	-1 -2 -6 -10
Frame 3	-1 -3 -7 -11
Frame 4	-1 -4 -8 -12

Como se ha comentado previamente, en el modo LD-P, el primer frame se codifica como Intra-frame y el resto como frames tipo P. Los frames P realizan la estimación de movimiento usando frames previamente codificados y decodificados como frames de referencia. En HEVC, los frames tipo P pueden utilizar múltiples frames de referencia. En este artículo, un GOP (group of pictures) consiste en 4 frames P donde cada uno de ellos puede usar hasta 4 frames de referencia. En la Tabla IV se muestran las referencias usadas por cada frame P de un GOP. Al comienzo de la codificación de la secuencia de vídeo, el primer frame P tiene solo una referencia disponible, el frame I (-1). El segundo, tercero y cuarto frame tipo P usarán hasta 2 frames de referencia. Sin embargo, tras codificar varios GOPs, los frames P dentro de los subsiguientes GOPs usaran todas las posibles 4 referencias. Los tiempos mostrados en la Tabla III, son los requeridos para realizar la estimación de movimiento de un CTU sobre un único frame de referencia. Por lo tanto, cuando se usen varios frames de referencia, ese proceso de estimación de movimiento se repetirá para cada uno de los frames de referencia usados, reduciéndose hasta en 4 veces el número de CTUs por segundo que se pueden procesar si se usasen 4 frames de referencia.

TABLA V
CÓMPUTO EN FRAMES POR SEGUNDO DEL MÓDULO HARDWARE IME A 200 MHz

resolución vídeo	CTU_32 CTUs	CTU_32 fps	4xCTU_32 fps	CTU_64 CTUs	CTU_64 fps
832x480	390	8.93	35.71	104	7.85
1920x1080	2040	1.70	6.83	510	1.60
2560x1600	4000	0.87	3.48	1000	0.82

Para poder compensar esta situación, hemos desarrollado un módulo SAD HEVC que contiene 4 módulos CTU_32, cada uno con su propio canal de DMA, y que es capaz de realizar el procesamiento de 4 CTUs de 32x32 a la vez. El poder realizar la estimación de movimiento sobre los cuatro posibles frames de referencia al mismo tiempo reduciría el tiempo de codificación total un 77% de media. La Tabla V muestra el número de frames por segundo (fps) que pueden procesarse para las cuatro resoluciones de vídeo evaluadas si se usase nuestro módulo IME. También se muestra en la Tabla V, el número de CTUs por frame a procesar dependiendo tanto de la resolución del vídeo como del tamaño máximo de CTU. Como se puede apreciar, usando el módulo 4xCTU_32, podemos procesar 4 veces más frames por segundo, logrando una codificación de tiempo real para la menor resolución de vídeo.

IV. CONCLUSIONES

En este trabajo hemos presentado una arquitectura hardware IME. Hemos analizado cómo influye

el tamaño de CTU máximo en el rendimiento del codificador HEVC tanto en R/D como en tiempo de procesamiento. Como se ha mostrado hay pequeñas diferencias en R/D, siendo el máximo incremento en BD-rate un 4.3% cuando el tamaño máximo de CTU es 32x32 para los vídeos de muy alta resolución. En cuanto a la complejidad, el tamaño del CTU tiene un gran impacto el el tiempo total de codificación, siendo mucho más rápido cuando usamos el tamaño de CTU de 32x32. Cabe indicar que el módulo de estimación de movimiento ocupa entre el 85% y el 97.8% del tiempo total de codificación, con lo que nuestro módulo hardware IME reducirá ese tiempo significativamente.

En cuanto a la evaluación de nuestro módulo hardware IME, hemos medido el tiempo requerido para las transferencias a través del DMA así como el tiempo necesario para los cálculos. Los resultados muestran que nuestro módulo puede procesar 13928 y 3263 CTUs por segundo para tamaños máximo de CTU 32x32 y 64x64, respectivamente trabajando a 200MHz. Se puede apreciar que el cuello de botella del sistema es el proceso de transferencia del DMA, necesitando más del 70% del tiempo total para el procesamiento de un CTU.

Además hemos presentado un módulo SAD HEVC para el tamaño de CTU 32x32 que contiene 4 módulos IME, cada uno con su propio canal de DMA. Este nuevo módulo es capaz de procesar 4 CTUs de 32x32 a la par. Con este nuevo módulo integrado en HEVC y para el modo de codificación LD-P podemos reducir el tiempo total de codificación un 77% por termino medio, porque este módulo es capaz de realizar la estimación de movimiento de todos los 4 posibles frames de referencia al mismo tiempo.

Como trabajo futuro, queremos reducir los tiempos de transferencia del DMA, reutilizando parte del área de búsqueda para el calculo del IME de CUs contiguos, transfiriendo únicamente los nuevos pixels de referencia necesarios en cada momento y por otra parte empaquetando en una palabra de 64 bits ocho pixels del área de referencia.

Agradecimientos

Este trabajo ha sido financiado por el Ministerio de Ciencia, Innovación y Universidades con referencia RTI2018-098156-B-C54 cofinanciado con fondos FEDER (MINECO/FEDER/UE).

Referencias

[1] B. Bross, W.J. Han, J.R. Ohm, G.J. Sullivan, Y-K Wang, and T. Wiegand, "High efficiency video coding (HEVC) text specification draft 10," *Document JCTVC-L1003 of JCT-VC, Geneva*, January 2013.

[2] ITU-T and ISO/IEC JTC 1, "Advanced video coding for generic audiovisual services," *ITU-T Rec. H.264 and ISO/IEC 14496-10 (AVC) version 16, 2012*, 2012.

[3] G.J. Sullivan, J.R. Ohm, W.J. Han, and T. Wiegand, "Overview of the high efficiency video coding (HEVC) standard," *Circuits and systems for Video Technology, IEEE Transactions on*, vol. 22, no. 12, pp. 1648 –1667, December 2012.

[4] F. Bossen, B. Bross, K. Suhring, and D. Flynn, "HEVC complexity and implementation analysis," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 22, no. 12, pp. 1685–1696, 2012.

[5] Ahmed Medhat, Ahmed Shalaby, Mohammed S Sayed, and Maha Elsabrouty, "A highly parallel sad architecture for motion estimation in hevc encoder," in *IEEE Asia Pacific Conference on Circuits and Systems (APCCAS'14)*, Ishigaki, Nov. 2014, pp. 280–283.

[6] J. Byun, Y. Jung, and J. Kim, "Design of integer motion estimator of hevc for asymmetric motion-partitioning mode and 4k-uhd," *Electronics Letters*, vol. 49, no. 18, pp. 1142–1143, 2013.

[7] Xu Yuan, Liu Jinsong, Gong Liwei, Zhang Zhi, and Robert K.F. Teng, "A high performance vlsi architecture for integer motion estimation in hevc," in *IEEE 10th International Conference on ASIC (ASICON'13)*, Shenzhen, Oct. 2013, pp. 1–4.

[8] Thomas D'huys, "Reconfigurable data flow engine for hevc motion estimation," in *IEEE International Conference on Image Processing (ICIP'14)*, Paris, Oct. 2014, pp. 1223–1227.

[9] Antonio Navarro Purnachand Nalluri, Luis Nero Alves, "High speed sad architectures for variable block size motion estimation in hevc video coding," in *IEEE International Conference on Image Processing (ICIP'14)*, Paris, Oct. 2014, pp. 1233–1237.

[10] Estefania Alcocer, Roberto Gutierrez, Otoniel Lopez-Granado, and Manuel P. Malumbres, "Design and implementation of an efficient hardware integer motion estimator for an hevc video encoder," *Journal of Real-Time Image Processing*, pp. 1–11, 2016.

[11] HEVC software repository HM–14.0 reference model, https://hevc.hhi.fraunhofer.de/trac/hevc/browser/tags-/HM-14.0, ," .

[12] I. Kim, J. Min, T. Lee, W. Han, and J. Park, "Block partitioning structure in the hevc standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, no. 12, pp. 1697–1706, Dec 2012.

[13] Frank Bossen, "Common test conditions and software reference configurations," Tech. Rep. JCTVC-L1100, Joint Collaborative Team on Video Coding, Geneva, January 2013.

[14] G. BjÃ¸ntegaard, "Document vceg-m33: Calculation of average psnr differences between rd-curves," Tech. Rep., ITU-T VCEG Meeting, Austin, Texas, USA, Tech. Rep, 2001.

Eficiencia energética en Algoritmos Genéticos

Josefa Díaz-Álvarez, Francisco Fernández de Vega, Juan Ángel García[1], Francisco Chávez[2] y Jorge Alvarado[3]

Resumen— **Al aplicar los Algoritmos Evolutivos (AEs) a problemas de optimización, dos de los elementos principales tomados en cuenta para evaluar su rendimiento son: la calidad de la función de coste y el tiempo de cómputo. Estos dos valores se utilizan para comparar el rendimiento de diferentes versiones de un algoritmo, comparar diferentes parámetros en la configuración de un único algoritmo, o incluso comparar un AE particular con otras heurísticas disponibles. Sin embargo, existe una nueva tendencia en ciencias de computación que intenta contextualizar estas características desde una nueva perspectiva: el consumo de energía. Este artículo presenta el análisis de un algoritmo genético estándar, utilizando dos problemas ampliamente conocidos, considerando tanto la calidad de la función de coste y el tiempo de cómputo, como también el consumo de energía empleado, calculado dicho consumo utilizando hardware alimentado por baterías para su ejecución. Los resultados muestran que alguno de los principales parámetros del algoritmo tienen impacto en el consumo instantáneo de energía, lo cual se desvía del comportamiento esperado, y por tanto, afecta a la cantidad de energía necesaria para ejecutar el algoritmo. Aunque estamos aún lejos de encontrar un camino para diseñar EAs eficientes energéticamente, pensamos que los resultados abren una nueva perspectiva que nos permitirá encontrar este objetivo en el futuro.**

Palabras clave— **Algoritmos Evolutivos, Eficiencia Energética, Algoritmos Genéticos, Consumo de Energía**

I. Introducción

Cuando se analizan los algoritmos evolutivos para evaluar su rendimiento, los investigadores consideran en primer lugar, la calidad de las soluciones obtenidas y después el tiempo necesario para obtener la solución. A pesar de que las versiones paralelas han sido implementadas para ahorrar tiempo de cómputo y se dispone de una gran cantidad de modelos estructurados y tecnologías hardware [1] para ello, los investigadores normalmente se refugian en versiones secuenciales y recurren a modelos paralelos cuando el problema a solucionar necesita días o semanas para alcanzar la solución.

En el análisis que presentamos en este trabajo nos centraremos en la versión secuencial del Algoritmo Genético, enfoque usado con mayor frecuencia, aunque el estudio se podría adaptar para versiones paralelas y distribuidas del algoritmo.

Han pasado cuatro décadas desde que los AGs fueron propuestos por J. Holland [2], y durante años esta heurística de optimización y búsqueda basada en la evolución ha sido ejecutada por los investigadores en multitud de dispositivos hardware, lo que les ha permitido obtener soluciones de calidad y en un tiempo menor. Sin embargo, el consumo de energía del dispositivo donde se ejecutaba el algoritmo nunca ha sido considerado como elemento de interés. En otras áreas de ciencias de computación el tema ya ha entrado en el terreno de la optimización [3], [4], [5], [6], y hasta hace muy poco tiempo, el único trabajo previo que vinculaba los AEs con el consumo de energía tenía como objetivo la optimización del consumo de energía en otras áreas [7].

Recientemente, algunos artículos han incluido el consumo de energía como uno de los problemas de interés a estudiar al analizar el comportamiento de los EAs [8], especialmente cuando se utilizan pequeños dispositivos alimentados por baterías para ejecutar este tipo de meta-heurísticas.

Durante los últimos años, hemos sido testigos de los primeros intentos de estudiar el comportamiento relacionado con el consumo de energía en los algoritmos evolutivos y se han publicado los primeros artículos sobre el tema [9], que incluyen algún análisis preliminar del consumo de energía asociado a diferentes plataformas hardware [10]. Este estudio es particularmente apropiado cuando se utilizan dispositivos alimentados por batería (tales como dispositivos móviles u ordenadores portátiles desconectados de la red eléctrica), por razones obvias. Sin embargo, que sepamos no se ha presentado ningún estudio específico que analice el impacto de la configuración del algoritmo en la energía consumida para alcanzar una solución.

Este artículo presenta, por primera vez, un análisis de este tipo para AGs. Aunque los resultados son todavía preliminares, consideramos que prepara el camino para una mejor comprensión del algoritmo bajo esta nueva perspectiva y permitirá en el futuro el diseño de AEs más eficientes desde el punto de vista energético.

El resto del artículo se organiza como sigue: en la sección II se presenta la motivación para realizar este análiss y una discusión sobre los resultados esperados. La sección III describe la metodología seleccionada para el análisis y los problemas de referencia. La sección IV muestra los resultados obtenidos y, por último, la sección V presenta las conclusiones y el trabajo futuro.

II. Considerando el consumo de energía en AGs

Como se ha mencionado con anterioridad, estamos interesados en analizar la huella energética de los algoritmos evolutivos, con especial atención a la ejecución de los mismos en dispositivos alimentados por

[1]Dpto. de Tecnología de los Computadores y de las Comunicaciones. Univ. de Extremadura. e-mail: mjdiaz,fcofdez,jangelgm@unex.es

[2]*Dpto.* de Sistemas Informáticos y Telemáticos. Univ. de Extremadura. e-mail: fchavez@unex.es

[3]Grupo GEA. Univ. de Extremadura. e-mail: jorgealvaradodiaz@gmail.com

baterías, tales como ordenadores portátiles, tabletas, teléfonos móviles, todos ellos con una dependencia fundamental de la energía disponible en sus baterías. Pero antes, debemos entender por qué este problema no se ha abordado con anterioridad y por qué pensamos que merece la pena abordarlo.

Consideremos primero la versión estándar del Algoritmo Genético, cuando lo aplicamos al problema *maxone*. Entre los principales parámetros del algoritmo y el problema a solucionar, podemos destacar el tamaño de cromosoma, número de individuos en la población y número de generaciones, por nombrar unos cuantos.

Así, el algoritmo tiene que repetir un conjunto de operaciones estándar durante el número de generaciones determinado: evaluación de la función de coste, selección, cruce y mutación; cuanto mayor sea el tamaño de la población, mayor será el número de repeticiones de estas operaciones.

Nos centraremos en la versión secuencial del algoritmo. Esto significa que una única CPU se dedicará a ejecutar el algoritmo y que las diferencias en tiempo de ejecución por las actividades del sistema operativo se pueden descartar utilizando los datos de un gran número de ejecuciones y calcular posteriormente el tiempo medio de ejecución. Actualmente, la literatura ha considerado muy poco la influencia del sistema operativo en el tiempo de ejecución de los algoritmos evolutivos, de manera más concreta al ejecutar modelos paralelos en una única CPU, dado que el sistema operativo debe gestionar la planificación de diferentes procesos ejecutándose simultáneamente en un único procesador [11].

Por tanto, empleamos el enfoque secuencial en el experimento teórico que describimos como nuestro punto de inicio y que en el futuro incluirá el sistema operativo en el panorama completo del análisis del consumo de energía.

A. Tamaño de población y consumo de energía

El número de generaciones, tamaño de población y funciones de coste son los componentes principales que influyen en el tiempo de ejecución de un Algoritmo Genético estándar. Aunque diferentes versiones de los operadores de cruce y mutación pueden suponer diferentes tiempos de cómputo, creemos que los parámetros descritos anteriormente son los que tienen una mayor influencia en el tiempo necesario para ejecutar el AG.

Si decidimos, por tanto, ejecutar el algoritmo durante N generaciones, el tiempo de ejecución será más corto que cuando ejecutamos $N+1$, y más largo que con $N-1$. No consideramos aquí si la solución se encuentra antes de completar el número de generaciones, lo cual se podría asegurar fácilmente haciendo el problema más dificil (por ejemplo, incrementando el tamaño del cromosoma en el problema Maxone). De forma similar, si utilizamos I individuos en la población, el tiempo de ejecución será más corto que al usar $I+1$, dado que para cada individuo se debe evaluar la función coste. Por último, un experimento con una función de coste que necesite más tiempo llevará a un tiempo de ejecución mayor.

Debemos tener en cuenta que cuando se abordan problemas reales, la solución perfecta generalmente no se encuentra y el algoritmo se configura para que pare al alcanzar un tiempo de ejecución máximo. En lo sucesivo, tomaremos este enfoque, afinando la dificultad del problema para que no se encuentre la solución durante el tiempo asignado.

B. Consumo de energía y tiempo de ejecución

Consideremos que la CPU, cuando ejecuta el algoritmo, dedica exactamente el mismo esfuerzo, independientemente de la operación que esté realizando. Si éste fuera el caso, significaría que el consumo instantáneo de energía es el mismo a lo largo del experimento y la energía total consumida se podría calcular fácilmente multiplicando el consumo instantáneo de energía (en cualquier momento durante el experimento), por el total de tiempo necesario para ejecutarlo. En consecuencia, se podría describir una relación lineal, vinculando el tiempo y el consumo de energía.

Este parece ser el caso para el área de investigación de AEs, donde implicitamente se asume la importancia del tiempo de cómputo y prescinde del estudio de la energía consumida, debido a su relación directa con el tiempo de cómputo.

Asumiendo las consideraciones descritas anteriormente, podríamos construir fácilmente una gráfica mostrando el consumo de energía estimado para valores diferentes de los parámetros en un AG. Por ejemplo, si una población con tamaño N consume una cantidad de energía E dada durante una ejecución, si ampliamos el tamaño de población en una serie de experimentos, esperamos tener un consumo de energía proporcional al incremento, debido a que el algoritmo realizará un número de operaciones adicionales (evaluaciones de la función de coste, mutaciones, cruce, ...) proporcionales al nuevo número de individuos en la población. En la figura 1, mostramos el comportamiento esperado. El resultado se ha obtenido utilizando valores para un único experimento y proyectándolos para el resto, dado que se espera el mismo comportamiento con tamaños de población mayores. También consideramos que la solución no se encuentra durante la ejecución, si no, un experimento pararía y la energía consumida sería distinta.

La principal idea para construir la figura 1 es que el consumo instantáneo de energía del procesador es constante independientemente de la operación realizada: cuanto mayor es el tiempo de ejecución de un experimento, mayor es la energía necesaria para ejecutarlo. Como se puede ver en la figura, el consumo de energía es proporcional al tamaño de población.

C. Dificultad del problema y tamaño del cromosoma

Un análisis similar se podría hacer para problemas donde la dificultad se refiere al tamaño del cromosoma. Por ejemplo, cuando se considera el problema

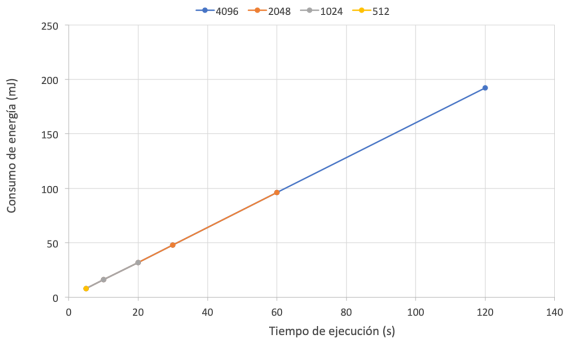


Fig. 1. Consumo de energía de un AG con diferentes tamaños de población

maxone, a mayor tamaño del cromosoma mayor será la dificultad y, por tanto, mayor también el tiempo de evaluación. Suponiendo una vez más que no hay diferencias en la energía instantánea consumida por la CPU en ninguno de los posibles experimentos que podemos lanzar, la energía total consumida solo dependerá del tiempo y se podría obtener un gráfico similar al mostrado en la figura 1 con distintos tamaños de cromosomas.

Nuestra hipótesis es que el comportamiento esperado es lo que impide que los investigadores lo tengan en cuenta al estudiar el comportamiento del Algoritmo Genético: si tiempo de ejecución y consumo de energía son valores proporcionales, no hay razón para estudiar ambos. Una vez que se obtiene el tiempo de ejecución, el consumo de energía se puede calcular fácilmente.

Sin embargo, se ha descrito recientemente que la energía es importante para decidir la plataforma hardware más eficiente para ejecutar un algoritmo. Cuando la eficiencia es clave, el consumo de energía es el punto de vista correcto para evaluar la plataforma hardware preferida [10].

En cualquier caso, ¿podríamos todavía suponer que energía y tiempo de ejecución son valores proporcionales de una entidad única sin evidencia experimental? ¿Influyen de alguna manera algunos de los principales parámetros del AG en la energía necesaria para ejecutar el algoritmo?

Creemos que tal análisis es útil para confirmar o descartar la hipótesis supuesta y es lo que tratamos en las siguientes secciones. Solo nos centramos en el consumo de energía en este análisis preliminar y no consideramos la calidad de las soluciones encontradas.

III. Metodología

La idea es, por tanto, estudiar si alguno de los parámetros principales de los AG tiene alguna influencia en la energía total consumida al ejecutar el algoritmo o, por el contrario, los valores de los parámetros no influyen en él.

A. Dispositivo alimentado por batería

Como se ha descrito anteriormente, estamos principalmente interesados en la eficiencia energética cuando se usan dispositivos alimentados por baterías, donde la energía disponible es limitada, aunque las conclusiones se podrían extender fácilmente a cualquier plataforma hardware.

En este primer estudio, hemos seleccionado una Tablet Lenovo Tab2, A10 - 70F con un Mediatek SoC MT8165, que es un modelo estándar representativo entre las opciones disponibles.

Esta tablet incorpora un MediaTek MT8165, 64-bit ARM-based SoC para dispositivos Android, y se lanzó en 2014. El procesador con cuatro núcleos a 1.5 GHz se fabricó en 28 nm y está basada en la arquitectura Cortex-A53. Además del núcleo de la CPU, integra una GPU ARM Mali-760 MP2 y un controlador de memoria LPDDR3 (32-bit, 800 MHz, 6.4 GB/s).

B. Medir el consumo de energía

Una vez seleccionado el dispositivo móvil, necesitamos una forma adecuada para medir el consumo de energía del algoritmo. Hemos optado por una aplicación que nos permita analizar el consumo real de la batería de cualquier dispositivo con Android: PowerTutor [12].

El consumo de energía de una determinada aplicación en ejecución se puede ver influido por los diferentes componentes hardware que utiliza. PowerTutor es una herramienta software libre, que permite monitorizar el consumo de energía de los diferentes componentes hardware que conforman un dispositivo móvil. Los componentes que PowerTutor mide son los siguientes: CPU, OLED/LCD, WIFI, 3G, Audio.

El sistema completo usado para medir el consumo de energía se compone de los siguientes elementos: (i) un servicio web y (ii) un plug-in para la propia aplicación PowerTutor. A continuación, describimos cada uno de los componentes del sistema.

B.1 Servicio web

El servicio web desarrollado incorpora una versión de un AG para probar el plug-in. El algoritmo se ejecuta en un sitio web, dentro del nuevo plug-in desarrollado para PowerTutor, para que podamos obtener fácilmente la energía que consume. El algoritmo necesita dos ficheros: un fichero *javascript* que contiene el AG y un fichero *html* que hace referencia a este fichero *javascript*.

El AG ha sido implementado usanto dos enfoques diferentes, con y sin librerías externas. En el primer enfoque, hemos codificado el AG desde Scratch. En el segundo, se han utilizado las librerías externas para su codificación. Hemos usado la librería NodEO [13], que incluye las funciones necesarias para crear un AG sencillo en JavaScript utilizando el formato CommonJS [1].

B.2 Plug-in para PowerTutor

Una vez implementado el servicio web, el algoritmo se puede ejecutar en cualquier dispositivo Android y

[1] http://requirejs.org/docs/commonjs.html

TABLA I
PARÁMETROS Y VALORES ANALIZADOS PARA EL PROBLEMA *Maxone*

Tamaños de población	32, 64, 128, 256,512, 1024 y 2048
Tamaños de cromosoma	32, 64, 128, 256, 512, 1024, ... 32768

TABLA II
PARÁMETROS Y VALORES ANALIZADOS PARA EL PROBLEMA *Trap*, CON TAMAÑO DE CROMOSOMA: 200

Tamaños de población	32, 64, 128, 256, 512 y 1024

podemos medir el consumo de energía mediante el plug-in de PowerTutor desarrollado.

Hemos creado un plug-in que añade una nueva funcionalidad a PowerTutor. Nuestro plug-in es capaz de ejecutar una tarea y medir el consumo de energía de la CPU resultante de la ejecución. El plug-in se encarga de monitorizar el entorno web donde el AG se ejecuta. El AG se debe codificar como un fichero JS y se ejecuta por el componente Webview de Android.

C. Problemas y parámetros evaluados

Para este estudio preliminar se han seleccionado dos conocidos problema de AG: los problemas *Maxone* y la función *Trap* [14].

La idea es tener diferentes configuraciones variando alguno de los principales parámetros de un AG, realizar ejecuciones largas de los experimentos y, posteriormente calcular el total de energía consumida con las configuraciones diferentes de los parámetros.

La versión generacional de un algoritmo se ejecutará con un límite máximo de tiempo establecido para la ejecución: 300 segundos. Las diferentes configuraciones que se han probado para confirmar alguna de las conclusiones presentadas, se han construido variando el tamaño de población y dificultad del problema (tamaño del cromosoma) para *Maxone*, y el tamaño de población para la función *Trap* con tamaño de cromosoma fijo.

Las tablas I y II sintetizan las pruebas realizadas. Por cada valor de los parámetros, se lanzan 30 ejecuciones independientes y se obtienen los valores promedios mostrados abajo. A pesar de la dificultad establecida para el problema, alguna de las ejecuciones con poblaciones grandes fueron capaces de encontrar una solución antes de los 300 segundos. En tales casos, el cálculo de los valores medios se realizó con las ejecuciones que alcanzaron ese tiempo.

IV. RESULTADOS

Comenzamos los experimentos con el problema *Maxone* probando con diferentes tamaños de cromosoma. Debemos tener en cuenta que para este problema particular, a mayor tamaño de cromosoma, se necesita más tiempo para encontrar una solución, debido a que la dificultad se incrementa con el tamaño. Fijamos el tamaño de población a 10, intentando evitar ejecuciones muy cortas y, así, asegurar que podemos obtener datos del consumo de energía para cada experimento a lo largo de los 300 segundos, fijados por ejecución. Aunque hay otras posibilidades, pensamos que este experimento suministra información importante para el objetivo que perseguimos.

Como podemos ver en la figura 2, siendo todas las curvas parecidas, aparecen algunas diferencias cuando probamos tamaños grandes de cromosomas. Además, alguna de las diferencias parecen desviarse de lo que podíamos esperar cuando analizamos las posibles anomalías: al usar 4096 bits, la energía total consumida es mayor que cuando usamos 16384 bits, aunque las diferencias son reducidas. Por otra parte, con el fin de apreciar mejor las diferencias con valores más pequeños para este parámetro utilizado, mostramos una versión ampliada de la parte más baja de la gráfica en la figura 3. De nuevo, podemos ver algunas diferencias, aunque demasiado reducidas. Ahora, la pauta relativa a las anomalías confirma lo que se vió anteriormente: algunas veces valores grandes para el tamaño del cromosoma consumen una cantidad de energía más pequeña en comparación con otros tamaños de cromosomas más pequeños.

Así vemos, que aunque el comportamiento considerando el tamaño del cromosoma no es muy diferente al esperado, descrito en la sección anterior y donde todas las curvas están unas encima de las otras, aparecen algunas anomalías que merecen ser estudiadas en el futuro.

A pesar de ello, decidimos realizar más experimentos utilizando diferentes valores para el tamaño de población (ver tabla I), y fijamos un valor grande para el tamaño de cromosoma (16384 bits) para asegurar que las pruebas se ejecutan durante 300 segundos.

En la figura 4 se muestran los resultados obtenidos. Podemos observar que la figura es completamente diferente de lo esperado (como se ve en la figura 1): en vez de estar todas las líneas superpuestas, el experimento muestra que hay diferencias importantes entre los tamaños de población en cuanto al consumo de energía del algoritmo. Especialmente importantes son las diferencias cuando se utilizan tamaños grandes de población: la energía consumida es menor en comparación con tamaños de población pequeños, lo cual es un resultado sorprendente e intrigante. Aunque se encontraron similares diferencias para los tamaños de cromosoma, las diferencias son mucho mayores al considerar el tamaño de población.

Decidimos, por tanto, centrar nuestras pruebas para el segundo problema, la función *Trap*, realizando una serie de ejecuciones con los tamaños de población mostrados en la tabla II. Los valores promedio de las 30 ejecuciones se muestran en la figura 5. En la figura 5 se visualiza una progresión de líneas, cada una con distinto comportamiento de consumo de energía.

Se observa una progresión de comportamientos casi perfecta: a mayor tamaño de población, mayor energía necesaria para la ejecución. Sin embargo,

la curva correspondiente al tamaño de población 512 no continúa la tendencia y describe un comportamiento del consumo de energía bastante distinto de las demás. Pero, debemos indicar que al utilizar tamaños grandes de población, algunas de las ejecuciones encontraron la solución al problema antes de los 300 segundos, por ello, los valores promedio en los últimos intervalos de tiempo incluyen un menor número de ejecuciones. En cualquier caso, esto no afectaría al comportamiento del consumo de energía, que se diferencia de nuevo del esperado.

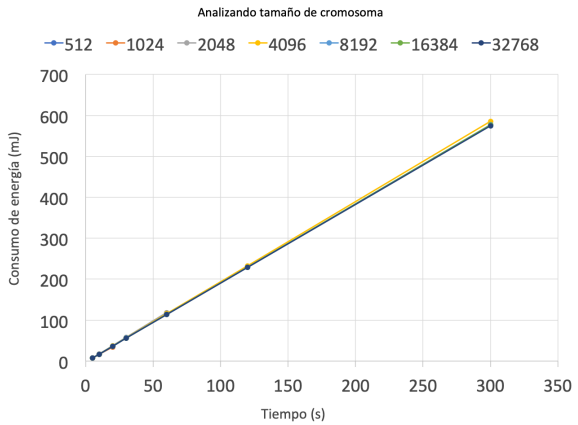


Fig. 2. Consumo de energía *Maxone*: Analizando distintos tamaños de cromosoma. Tamaño de población = 10

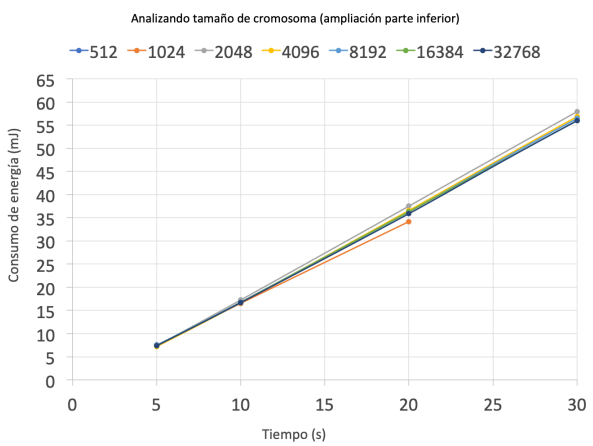


Fig. 3. Consumo de energía *Maxone*: Analizando distintos tamaños de cromosoma (ampliación). Tamaño de población = 10

Los resultados mostrados arriba corresponden a los datos obtenidos con el AG en ejecución. Tenemos también que medir qué sucede cuando la tablet está encendida y el algoritmo no se ha lanzado. Por supuesto que el sistema operativo tiene su papel y algunos dispositivos consumirán más energia, pero esta situación es similar a cuando ningún programa se está ejecutando. Por ello, medir el consumo de energía en este caso concreto ayudará a contextualizar los resultados anteriores.

Por tanto, para realizar una adecuada comparación, se ha eliminado del AG todas las sentencias de ejecución, para que no se realize ninguna instrucción durante una hora (se ha creado un algoritmo AG "vacío"). Una vez más, se han lanzado 30 ejecuciones independientes y se han calculado los valores promedio: 30 operaciones de inicialización de la tablet seguidas por la ejecución del algoritmo "vacío" y se midió el consumo de energía.

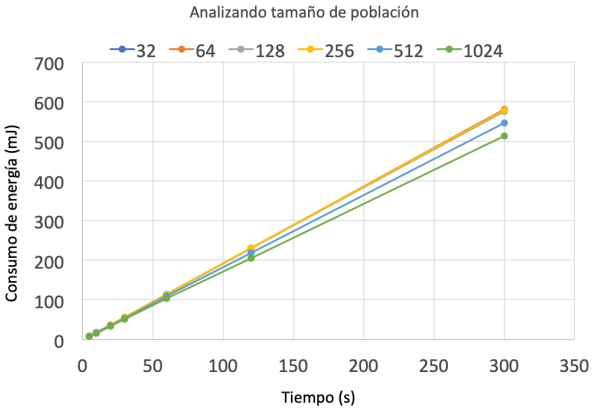


Fig. 4. Consumo de energía *Maxone*: analizando distintos tamaños de población (tamaño de cromosoma = 16384)

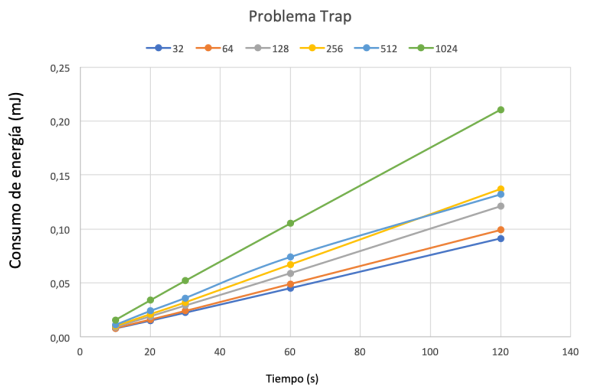


Fig. 5. Consumo de energía *Trap*: analizando distintos tamaños de población.

La figura 6 muestra los resultados. Si nos centramos en la marca 300, el máximo tiempo asignado para los experimentos anteriormente explicados, vemos que el consumo total de energía es menos de 300mJ, y esto se puede comparar con los resultados en la figura 4, que muestra valores superiores a 500mJ para *Maxone*.

La tabla III presenta los datos para el problema *Maxone* con los distintos tamaños de cromosomas probados, 10 individuos por población. Cada columna contiene media del consumpo de energía/desviación estándar de las 30 ejecuciones en los diferentes intervalos de tiempo. Los valores vacíos en algunas de las celdas son debidos al corto tiempo de ejecución (los experimentos finalizan antes de alcanzar algún valor de tiempo). Como podemos observar, cuando empleamos el tamaño de cromosoma de 4096 (y 2048 cuando se dispone de datos), la energía consumida es mayor que al utilizar valores superiores a uno después de 30 segundos. Podemos verificar, por ejemplo, el valor obtenido después de 300 segundos: 585.5 (desviación estándar de 4.9), mientras que para cromosomas más largos el consumo de energía

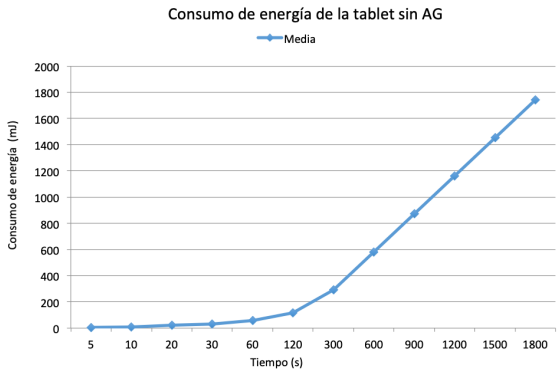


Fig. 6. Energía consumida por el dispositivo tablet cuando el AG no se está ejecutando.

es alrededor de 577. Esto confirma lo que habíamos visto en las figuras anteriormente mostradas: que existen diferencias significativas en los patrones de consumo de energía influenciados por la configuración de los parámetros.

Por otra parte, la tabla IV incluye información similar para el problema *Maxone* utilizando tamaños diferentes de población con un tamaño de cromosoma previamente fijado: 16384 bits. Para este conjunto de experimentos, se puede obsevar que cuando se utilizan grandes poblaciones (512 y 1024), la desviación estándar permite afirmar con confianza que el consumo de energía es más pequeño, un comportamiento que se observó en la figura 4 y que necesita más investigación en el futuro.

La tabla V detalla información parecida para el problema *Trap* con diferentes tamaños de población y un tamaño de cromosoma de 200 bits. Los resultados permiten otra vez confiar en las diferencias encontradas. La última fila en las tres tablas anteriores presenta los valores medios y la desviación estándar, valores calculados sobre el conjunto completo de pruebas para cada marca de tiempo fijada.

En conjunto, después de esta serie de experimentos podemos confirmar que por primera vez hemos detectado que algunos valores de los principales parámetros de un AG influyen en el consumo de energía; especialmente relevante es el tamaño de población utilizado. Esto podría deberse a los patrones de uso de la memoria, que afecta a las operaciones de acceso a la memoria cache. Pero es necesario realizar más pruebas para confirmar plenamente y entender si es un comportamiento general de los EAs y que una configuración adecuada de los parámetros puede permitirnos diseñar AGs energéticamente más eficientes.

V. Conclusiones

A pesar de que tradicionalmente la calidad de la función de coste y el tiempo necesario para alcanzar una solución han sido los principales elementos para analizar el comportamiento de los algoritmos, este trabajo propone y describe la razón para considerar el consumo de energía como una nueva medida a aplicar cuando se analiza el comportamiento de los AGs.

Pocos estudios han incluido el consumo de energía como tema de interés, probablemente debido a la aceptación de la relación lineal entre el tiempo de cómputo y el consumo de energía.

Este artículo presenta un análisis preliminar sobre la influencia de alguno de los principales parámetros de un AG en sus patrones de consumo de energía. Hasta donde sabemos, este análisis es el primero en estudiar el impacto de la configuración de los parámetros de un AG en la energía requerida para ejecutar el algoritmo.

Para el análisis se han seleccionado dos problemas bien conocidos: el problema *Maxone* y la función *Trap*. Se han evaluado distintos tamaños de población y de cromosoma, y en cada configuración se ha determinado la energía necesaria para ejecutar el algoritmo durante 300 segundos, comparando dichos valores con los resultados teóricamente esperados (relación lineal entre el tiempo y la energía).

En ambos problemas, se han encontrado diferencias con respecto al comportamiento esperado: (i) la correlación entre tiempo y energía no es lineal; (ii) existe una conexión entre los valores de los parámetros y el consumo de energía. En cualquier caso, las anomalías encontradas, tales como consumo menor de energía con grandes tamaños de población, merece más investigación si nuestro objetivo es encontrar soluciones con una reducción en el consumo de energía.

Aunque los experimentos se han ejecutado en un dispositivo Android alimentado por batería, esperamos extender el estudio en el futuro a un mayor conjunto de dispositivos, incluyendo ordenadores portátiles, dispositivo Raspberry pi, etc, de modo que podamos confirmar este comportamiento independientemente del hardware subyacente y, así, ser capaz de diseñar algoritmos evolutivos energéticamente eficientes en el futuro.

Agradecimientos

Agradecemos el apoyo del Ministerio de Economía y Competitividad de España en el proyecto TIN2017-85727-C4-{2,4}-P, Gobierno Regional de Extremadura, consejería de Comercio y Economía, el Fondo Europeo de Desarrollo Regional, una forma de construir Europa, en el proyecto IB16035 y Junta de Extremadura, proyecto GR15068.

Referencias

[1] F. Fernández de Vega, J. I. Hidalgo Pérez, and J. Lanchares, *Parallel Architectures and Bioinspired Algorithms*, John Wiley & Sons, Ltd, 2010.

[2] J. H. Holland, *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*, 1975.

[3] Mao Ye, Chengfa Li, Guihai Chen, and J. Wu, "Eecs: an energy efficient clustering scheme in wireless sensor networks," in *PCCC 2005. 24th IEEE International Performance, Computing, and Communications Conference, 2005.*, 2005, pp. 535–540.

[4] Tiago Camilo, Carlos Carreto, Jorge Sá Silva, and Fernando Boavida, "An energy-efficient ant-based routing algorithm for wireless sensor networks," in *Ant Colony Optimization and Swarm Intelligence*. 2006, pp. 49–59, Springer Berlin Heidelberg.

TABLA III

CONSUMO DE ENERGÍA *Maxone*: ANALIZANDO DISTINTOS TAMAÑOS DE CROMOSOMA. TAMAÑO DE POBLACIÓN = 10

	Tiempo (s)						
Tamaño de cromosoma	5	10	20	30	60	120	300
512	7.600/**0.44**						
1024	7.450/**0.36**	16.532/**0.49**	34.187/**0.61**				
2048	7.326/**0.53**	17.256/**0.62**	37.528/**1.04**	57.931/**1.51**	118.477/**1.60**		
4096	7.147/**0.51**	16.676/**0.66**	36.592/**0.94**	56.845/**1.01**	115.755/**1.55**	232.705/**3.27**	585.578/**4.90**
8192	7.456/**0.28**	16.791/**0.63**	36.315/**0.92**	56.435/**1.87**	113.782/**2.02**	229.485/**2.39**	577.026/**4.06**
16384	7.377/**0.36**	16.730/**0.62**	36.307/**0.85**	55.975/**1.57**	113.836/**1.84**	229.275/**2.65**	576.852/**3.73**
32768	7.344/**0.33**	16.678/**0.48**	35.897/**0.78**	55.975/**1.44**	113.472/**1.76**	228.783/**2.66**	574.544/**7.13**
Valor medio/Desviación estándar	**7.386/0.425**	**16.777/0.623**	**36.354/1.195**	**56.632/1.652**	**115.041/2.562**	**230.040/3.128**	**578.256/6.504**

TABLA IV

CONSUMO DE ENERGÍA *Maxone*: ANALIZANDO DISTINTOS TAMAÑOS DE POBLACIÓN. TAMAÑO DE CROMOSOMA = 16384

	Tiempo (s)						
Tamaño de población	5	10	20	30	60	120	300
32	7.334/**0.36**	16.487/**0.63**	36.056/**0.80**	55.761/**1.06**	113.828/**1.81**	229.618/**2.50**	576.326/**4.79**
64	7.295/**0.44**	16.755/**0.78**	35.767/**1.00**	55.994/**1.66**	113.972/**2.19**	230.513/**3.27**	580.024/**4.57**
128	7.353/**0.46**	16.791/**0.86**	36.383/**1.41**	56.016/**1.83**	113.738/**2.76**	229.958/**4.29**	577.435/**8.27**
256	7.401/**0.42**	16.902/**0.62**	36.089/**0.99**	55.552/**0.90**	114.025/**1.70**	229.713/**2.75**	576.974/**4.56**
512	7.415/**0.29**	16.528/**0.57**	34.473/**0.98**	53.178/**1.25**	108.981/**1.69**	217.755/**2.71**	546.043/**4.48**
1024	7.427/**0.23**	15.994/**0.57**	32.953/**0.90**	51.277/**0.99**	102.991/**1.57**	205.073/**1.91**	513.900/**2.95**
Valor medio	**7.371/0.37**	**16.576/0.73**	**35.287/1.58**	**54.630/2.22**	**111.256/4.57**	**223.772/9.95**	**561.784/24.96**

TABLA V

ENERGÍA CONSUMIDA *Trap*: ANALIZANDO DISTINTOS TAMAÑOS DE POBLACIÓN. TAMAÑO DE CROMOSOMA = 200

	Tiempo (s)					
Tamaño de población	10	20	30	60	120	300
32	7.63/**0.67**	14.95/**0.63**	22.36/**0.73**	45.14/**1.67**	91.18/**3.37**	195.95/**65.14**
64	8.06/**0.66**	16.18/**0.78**	24.45/**0.84**	48.90/**1.14**	99.35/**3.19**	238.89/**49.02**
128	9.3/**1.01**	18.74/**1.01**	28.94/**1.34**	59.18/**1.77**	121.50/**2.09**	310.71/**5.46**
256	10.47/**0.99**	21.16/**1.22**	32.25/**1.29**	66.73/**2.67**	136.64/**5.59**	342.17/**42.10**
512	11.48/**1.01**	23.79/**1.54**	35.96/**1.24**	73.78/**8.20**	131.64/**40.39**	220.00/**142.89**
Valor medio/Desviación estándar	**10.413/1.668**	**21.398/3.353**	**31.645/9.360**	**105.098/16.990**	**116.368/27.049**	**260.954/91.138**

[5] S. Albers, "Energy-efficient algorithms.," in *Communications of the ACM,53(5)*, 2010, pp. 86–96.

[6] W. R. Heinzelman, A. Chandrakasan, and H. Balakrishnan, "Energy-efficient communication protocol for wireless microsensor networks," in *Proceedings of the 33rd Annual Hawaii International Conference on System Sciences*, 2000, p. 10 pp. vol.2.

[7] María José Gacto, Rafael Alcalá, and Francisco Herrera, "A multi-objective evolutionary algorithm for an effective tuning of fuzzy logic controllers in heating, ventilating and air conditioning systems," *Applied Intelligence*, vol. 36, no. 2, pp. 330–347, Mar 2012.

[8] Camacho D., Lara-Cabrera R., Merelo-Guervós J.J., Castillo P. A., Cotta C., Fernández-Leiva A. J., Fernández de Vega F., and Chávez F., "From ephemeral computing to deep bioinspired algorithms: New trends and applications.," *Future Generation Computer Systems*, vol. 88, pp. 735–746, 2018.

[9] J. D. Álvarez, F. Chávez, P. A. Castillo, J. A. García, F. J. Rodríguez, and F. Fernández de Vega, "A fuzzy rule-based system to predict energy consumption of genetic programming algorithms.," *Computer Science & Information Systems, 15(3).*, 2018.

[10] Fernández de Vega, Chávez F. Díaz J. F., García J. A., P. A. Castillo, J. J. Merelo, and C. Cotta, "A cross-platform assessment of energy consumption in evolutionary algorithms.," in *In International Conference on Parallel Problem Solving from Nature*. September 2016, pp. 548–557, Springer, Cham.

[11] Francisco Fernández, G. Galeano, and J.A." Gómez, "Comparing synchronous and asynchronous parallel and distributed genetic programming models," in *Genetic Programming*. 2002, pp. 326–335, Springer Berlin Heidelberg.

[12] Z. Yang, "Powertutor-a power monitor for android-based mobile platforms," 2012.

[13] Juan-Julián Merelo, P. Castillo, A. Mora, A. Esparcia-Alcázar, and V. Rivas-Santos, "Nodeo, a multi-paradigm distributed evolutionary algorithm platform in javascript," in *Proceedings of the Companion Publication of the 2014 Annual Conference on Genetic and Evolutionary Computation*, 2014, pp. 1155–1162.

[14] Deb K. and Goldberg D. E., "Analyzing deception in trap functions," in *Foundations of Genetic Algorithms*, vol. 2, pp. 93–108. Elsevier, 1993.

Finding energy efficient hardware configurations under a power cap

Alberto Cabrera [1], Francisco Almeida [2],
Vicente Blanco [3] and Dagoberto Castellanos-Nieves[4]

Resumen— **The growing demand for more computational resources has caused an increased energy consumption in high performance computing infrastructures. To reach exascale systems, power and energy consumption have to be considered as a constraint to execute applications. Modern architectures are starting to provide tools that allow to manage directly the power constraints of a system. We perform an analysis of the performance and the energy efficiency using power cap technologies in various applications. We provide use cases for the Intel power cap and the Nvidia power limit. In our experimentation, we extract a pareto front of configurations that contain solutions, from most efficient energy usage to best possible performance.**

Palabras clave— **Energy Efficiency; Energy Aware Computing; Power Capping**

I. Introduction

Energy and power are a present problem of the high performance computing (HPC) community. A highly parallel system involves a very complex environment where efficiency affects a broad range of scopes. Data centers have been facing high power demands due to the high energy consumption of these machines. The current top systems in the Top500 list [1] have a power draw in the order of MW, with two of them closer to 20 MW. Building infrastructures, cooling systems, intercommunication networks, high performance and long term storage, computational hardware, scientific software and schedulers are examples of all the gears that need to fit and work together.

For the computational hardware, more efficient architectures have been introduced over the last years. Heterogeneous systems are more common, and there are efforts towards shifting to low power processors using highly efficient co–processors, as is the case of Nvidia general purpose graphic processing units (GPGPU), and the now discontinued Knights Landing Intel architecture.

A new possibility, power capping, is now considered in order to limit the power draw of computational elements. The aforementioned architectures provide tools to easily define these power constraints. This new possibility in the paradigm of HPC influence how programmers have to approach application development and how the environment is configured at runtime. Power capping is specially difficult in a heterogeneous system as it adds complexity to the parallel environment. To be able to use resources optimally, we need to increase our knowledge of the energetic behavior in these architectures.

Analytical modeling is a technique designed to study and provide knowledge of complex systems. Models give insight to the complexity of the computational environments and facilitate decision making to achieve our goals. It is possible to define different objectives depending on the environment, such as minimizing time to solution, reducing energy consumption, or finding the best performance under a power constraint.

We present an analysis of the performance and energy consumption in a given system when executing different applications. With this analysis, we extract a set of power cap configurations of optimal solutions. The obtained constraints allow, at runtime, to define a policy aimed to fulfill a power or performance goal. The illustrate a first case with a comparative for multiple applications using the same hardware and discuss the different optimal solutions with this configuration. In a second case, we analyze the situation for multiple hardware configurations for a single application. This case also clarifies the analysis for software services that have an undefined life cycle.

This paper is structured as follows: Section II covers the related work in the field of modeling and power savings in HPC. Section III describes our approach to analyze the energy usage and performance of our target application. Section IV illustrates our experimental setup and the computational experience applying this methodology in CPU and GPU environments. Finally, Section V summarizes our conclusions and future work.

II. Related Work

Traditionally in the HPC community, energy efficiency was improved by reducing the execution time in parallel applications. Plenty of libraries implement dense linear algebra packages and make use of computational models to achieve this goal. Plasma [2] and FLAME [3] are examples of libraries that optimize the computational resources for shared memory codes. Similarly in heterogeneous architectures composed by multicores and GPUs are present in MAGMA [4] and FlameGPU [5].

Analytical and regression modeling has been also applied widely to predict energy consumption and power draw for different applications. Performance

[1]HPC Group. Escuela Superior de Ingeniería y Tecnología Universidad de La Laguna, ULL. La Laguna. 38200 Tenerife. Spain, e-mail: Alberto.Cabrera@ull.edu.es.
[2]e-mail: falmeida@ull.edu.es.
[3]e-mail: Vicente.Blanco@ull.edu.es.
[4]e-mail: dcastell@ull.edu.es.

counter based models have been developed due to the impossibility of measuring energy in very short time windows. Multiple system components are observed through executions of different algorithms to correlate the power required by each component to these counters [6]. Tools, such as a system monitor interface [7], have been developed to supply accurate component power information. Our work differs in the usage of modeling as we do it at the application level using external measurements. Time constraints are also less strict, hence the possibility of simplifying the analysis.

Dynamic voltage and frequency scaling (DVFS) is also a common approach to improve the energy efficiency of applications. Outside of HPC environments, energy optimization is also a relevant topic. In cloud computing, energy–aware scheduling algorithms [8], [9] adjust the frequency of the host servers without degradation of the service performance. In HPC, methodologies and tools are developed using models with the help of DVFS, reducing the overall energy consumption while also improving performance for scientific applications [10], [11]. *E–AMOM* [12] is an example of a framework developed upon these methodologies, using models, DVFS, and dynamic concurrency throttling (DCT) to reach these goals.

However, the increasing importance of energy efficiency and power budgets have led to the development of more specialized tools. The interest in these tools is increasing as their usage relates directly to power limitations instead of affecting other parameters such as frequency. These tools have been proven valid as a possible replacement for the DVFS techniques present in the literature [13]. Using power capping technologies through the RAPL interface, *Conductor* [14] was developed as a middleware for post Sandy Bridge Intel architectures. It is capable of improving the performance of applications that are under a power budget allocating and shifting the power of the application.

Our methodology gives multiple possibilities for each architecture to offer a different points of energy savings, instead of choosing a unique energetic policy. This would prove useful for environments where performance can only be sacrificed partially or where limits can change with time.

III. Evaluating energy efficiency

In the Intel architectures used in this work, power limits can be assigned to different zones of the processors, known as power planes (PP). These PP are, for each package, the cores, the uncore and the DRAM. The power capping can be also applied to the package itself and, depending on the architecture, to all packages at the same time (PSYS). These limits are applied through the Intel RAPL interface and are not a strict power limit. The user specifies limitations in time intervals, known as *time windows*, and sets the average maximum power for the specified window. Then, the processor ensures the average power never surpasses the specified limits.

Nvidia, on the other hand, allows to set a power limit through their drivers to the system GPUs. However, these cards have great limitations on the minimum power needed by the GPU and offer less control when limiting the power consumption. In this case, a strict power constraint is defined that the hardware cannot surpass under any condition. High–end cards of the Tesla and Volta architectures offer better control over this energetic limitation.

We defined a simple benchmarking process to determine the behavior of a given target architectures. We learn the effects of applying power budgets to our heterogeneous system by executing small and simple instances of software. This limits our knowledge to the hardware/software combinations that are chosen for the study. However, experimentation is relatively simple, allowing to extend the software pool or the available hardware without complex efforts. We can formally define experimentation with the following parameters:

- H, set of architectures in our experimental environment.
- S, set of software to evaluate under power constraints.
- h, a computational node of our environment.
- s, a specific application or computational kernel.
- $P_{h,s}$, set of power configurations that can be applied to a given s and h. Depending on the user needs, examples of $P_{h,s}$ are:
  - A set where $|P_{h,s}| = 2$. These configurations are for best performance and for best efficiency.
  - A pareto optimal front containing every possible non–dominated power configurations.
  - A sub–set of the pareto front containing all power configurations under a user–defined power cap.
- p, power cap configuration. As mentioned earlier, Intel architectures require to define a 2–tuple of average power and a time window, i.e. $p = (avgpower, timewindow)$. For Nvidia GPUs a single maximum power value is needed, i.e. $p = (maxpower)$.
- $M_{h,s}$, hardware metrics gathered during the experimentation. Every element of $M_{h,s}$ is a 3–tuple of $(p, time, energy)$.

Algorithm 1 summarizes the experimental steps followed to analyze how power capping affects a given software and hardware. It is a generalized benchmark that can be applied to different applications in various platforms. The process can be summarized as performing multiple executions for various power cap configurations in target architecture. Once the experimentation is over, every pair h, s of hardware and software has have an associated $P_{h,s}$ to regulate energy and performance depending on the user needs.

Algorithm 1 Benchmark process

input: $H \rightarrow$ Hardware Pool, $S \rightarrow$ Software Pool
output: $P \rightarrow$ set of power settings

$P \leftarrow \varnothing$
for all $h \in H$ **do**
  $p_{min}, p_{max} = determine\_min\_max\_power(h)$
  $P_{exp} \leftarrow experimental\_interval(p_{min}, p_{max})$
    ▷ P_{exp}: Power experimental parameters
  for all $s \in S$ **do**
    $M_{h,s} \leftarrow \varnothing$ ▷ Hardware Metrics
    for $p \in P_{exp}$ **do**
      $t, e \leftarrow measure\_execution(h, s, p)$
      $M_{h,s} \leftarrow M_{h,s} \bigcup \{[p, t, e]\}$
    $P_{h,s} \leftarrow determine\_power\_settings(M_{h,s})$
    $P \leftarrow P \bigcup P_{h,s}$
return P

IV. Computational Experience

All hardware used in experimentation is defined in Tables I and II. Every node has a Debian 9 with kernel version 4.9.0–2–amd64 and GCC 4.8.5 installed. The nodes that have a Nvidia GPU have a Xeon E3–1505M CPU. Energy measurements were gathered using the appropriate EML [15] device driver in each case. As experiment have small differences from each other, more details are given in their specific sections.

CPU Name	# Cores	Base Freq.	TDP
i5-6200U	2	2.30 GHz	15 W
Xeon E3-1505M	4	2.80 GHz	45 W
Xeon D-1548	8	2.00 GHz	45 W
Xeon D-1577	16	1.30 GHz	45 W

TABLA I: Intel CPU power constraints

GPU Name	Min. Power	TDP
Tesla P40	125 W	250 W
Volta V100	100 W	250 W

TABLA II: Nvidia GPU power constraints

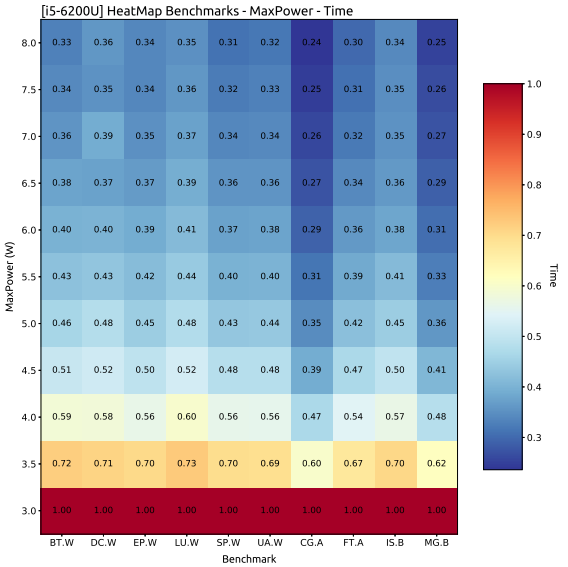


Fig. 1: NPB Benchmarks performance while applying Intel power cap. Lower values are better.

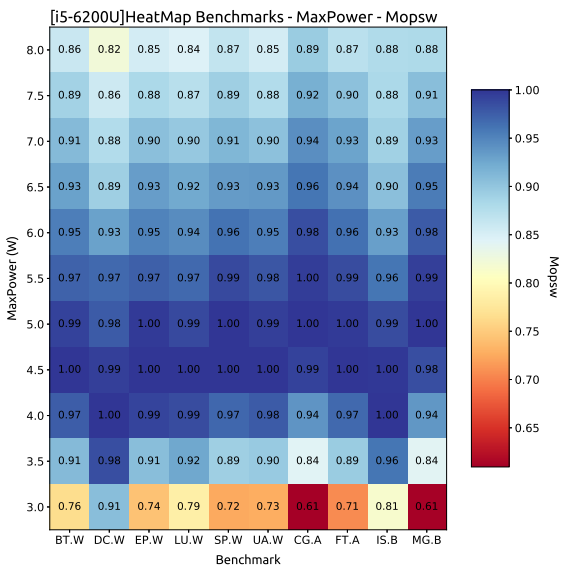


Fig. 2: NPB Benchmarks energy efficiency while applying Intel power cap. Higher values are better.

A. NPB Benchmarks

As a first approach, we present a use case of our methodology using the serial version 3.3.1 of the NAS Parallel Benchmarks (NPB) [16]. We performed the experimentation using an Intel i5-6200U CPU, a Skylake processor with power cap capabilities. Using the RAPL interface, we set multiple time windows for each specified power average to the CPU. As we have approached the problem with simplicity in mind, we limited the power consumption of the whole CPU, using the *PACKAGE* zone. Energy measurements were gathered using through the same RAPL interface using EML.

Figures 1 and 2 depict in a heatmap the behavior of every tested NPB kernel. The X axis represents each of the executed kernels from the NPB labeled as $AA.B$, with AA as the standard abbreviation of the kernel name and B as the size, or class, of the problem. These labels follow standard nomenclature provided from the NPB. The Y axis represents the maximum power allowed to the CPU, and each power have an average for several experiments using different time windows. Each column of the heatmap is normalized, with 1.00 being the highest value measured for each kernel. In both figures, dark blue represents the best values, while bright red illustrates the opposite. This translates to the lowest values representing the best performances in Figure 1 and the opposite for the efficiencies in 2.

The selected CPU allows a broader range for power

capping, below 3 W and over 8 W. However, values over 8 W were never reached, so they represent the absence of power capping. On the other side, metrics from the experimentation with a 3 W limit had shown an extreme decrement in both performance and efficiency.

Figure 1 illustrates the normalized performance for all kernels. The fastest configuration to solve the kernel is always obtained by removing the power limit, i.e. setting no limits for power consumption. Figure 2 depicts the same executions from the point of view of energy efficiency, measured in Mega operations per Watt (Mops/W), and then normalized for each case.

Three different patterns can be observed in the data:

- The *Data Cube* (DC) kernel reaches the best efficiency at 4 W, with 3.5 and 4.5 W obtaining good efficiencies as well. This kernel is memory intensive, and accesses the drive, in this case an SSD.
- The *Multi-Grid on a sequence of meshes* (MG) and the Conjugate Gradient (CG) reach the best efficiency at 5W. These kernels are also data bound, with CG performing irregular memory accesses.
- The rest of the kernels reach the best efficiency at 4.5 W and with the best performance at 8 W. These kernels are compute intensive, as the communication aspect is removed from the serial execution.

For each kernel, we extract a set with multiple optimal power cap configurations, with various efficiencies and performances. An specific example is given in the next subsection.

B. Tensorflow Image Classifier

The methodology requires slight modifications for measuring continuous applications, such as web servers, that do not have a finite life cycle and time–to–solution is not calculable. Our use case is an image classification software that has an undetermined cycle. This python application runs as an anaconda server that gathers images from a large provided database and trains an image classifier using a shared–memory implementation with Tensorflow. In this case, we will use both power capping technologies, the Intel power cap and the Nvidia power limit.

We will analyze the CPU case and the GPU case, taking into consideration the application of power limits to the CPU while executing the GPU code. In this case, EML gathers measurements from an external sensor that returns the energy consumption using an averaged power draw for the whole node. In the presented CPU cases, the PCI-express energy consumption is not taken into account, to remove the base power consumption of the GPU node.

Figure 3a presents on two curves the performance and the efficiency of the systems. This metric is the equivalent of the mega operations per Watt utilized in the NPB. The X axis represents the maximum power allowed for the CPU, the left axis the number of operations per Watt, represented using blue *o*, and the right axis the performance of the application in images per second, represented using orange +.

In Figure 1, we had an overview for multiple algorithms at the same time. Here we see a more in depth view of the performance and efficiency of the application for the different power capping settings. For each power limit in the X axis, several time windows were applied resulting in the multiple differentiated data values. However, as expected, we can observe that the time window is the least important of both factors when affecting efficiency.

Figure 3b puts together the performance and efficiency for each solution, allowing us to display the pareto front of non–dominated solutions. In this Figure, the X axis represents the performance of the classifier and the Y axis depicts its energy efficiency. This implies the best solutions are those higher and further to the right of the chart. Thus, our pareto front is the set of solutions united by the line in the top right of the plot.

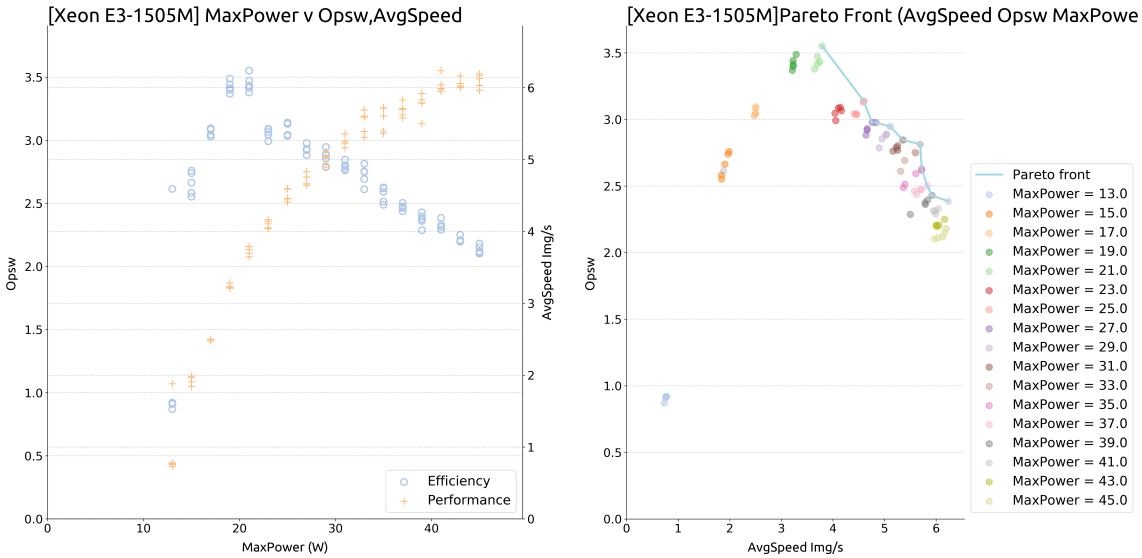
The specific values for the power cap are not included in a legend, however every different average power specified is depicted with a different color in this chart. It helps identify if the time window has more impact a given application, if a value for a given averaged power is not clustered.

Once executed in every CPU, the pareto front contains solutions in between the following power cap limits:

1. Xeon D-1548 best efficiency at 24W, and best performance at 45W.
2. Xeon D-1577 best efficiency at 22W, and best performance at 38W.
3. Xeon E3-1505M, the node illustrated in Figure 3, best efficiency at 21W, and best performance at 41W.

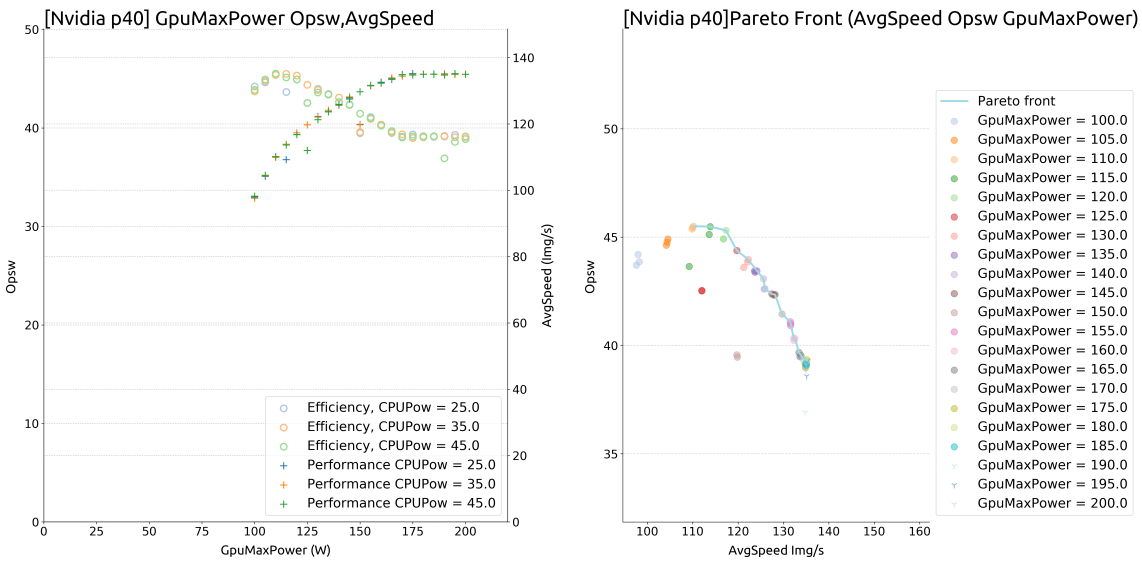
Finally, the same experimentation is replicated using the GPUs from Table II and the Nvidia power limit. Figures 4a and 4b represent the same data as the CPU counterpart. These nodes have a Xeon E3–1505M processor, the same CPU illustrated in the previous Figures.

In Figure 4a, additional information was added to illustrate both power cap limitations. The X axis represents the maximum power allowed for the GPU. Performance is differentiated from efficiency using different markers: + is used for performance, while *o* is used for efficiency. The left axis belongs to the performance depicted by the + markers, in number of operations per Watt. Similar data is presented for the performance of the application, with the right axis and the *o* markers. For both metrics, each color represent an specific power constraint for the CPU. However, they are difficult to differentiate because these CPU limits are already very high for the negligible CPU usage, and they do not affect performance



(a) Performance and Efficiency (b) Pareto front

Fig. 3: Xeon E3-1505M, Image classification analysis



(a) Performance and Efficiency (b) Pareto front

Fig. 4: Nvidia V100, Image classification analysis

nor efficiency.

Likewise, a pareto front can be extracted from this GPU executions, as shown in Figure 4b. Be aware that the axis limits in this chart have been adapted to visualize the data better.

From the hardware studied from Table II the following pareto front limits are extracted:

1. The Nvidia P40 has the best efficiency at 125W and the best performance starts at 175W.
2. The Nvidia V100 has the best efficiency at 115W and the best performance starts at 165W.

It is important to remark that these two cases could be studied more carefully, as the CPU will affect performance and efficiency if the power cap is applied more aggressively.

V. Conclusion

We have presented a simple methodology to analyze the effect of power capping for different architectures. As the behavior depends on both hardware and software, we have illustrated two use–cases. In one case we analyze different applications for a given hardware, and in the second case, we analyze the same software over a variety of hardware configurations. It is possible to extract a pareto front of solutions in order to take decisions that affect the performance and the energy efficiency of a given application. Obtaining this set also gives the possibility to maximize performance for a given power budget. In the future, we intend to extend the methodology to include the decision making to reduce the power

and energy usage of applications.

Acknowledgment

This work was supported by the Spanish Ministry of Education and Science through the TIN2016-78919-R project, the Government of the Canary Islands, with the project ProID2017010130 and the grant TESIS2017010134, which is co-financed by the Ministry of Economy, Industry, Commerce and Knowledge of Canary Islands and the European Social Funds (ESF), operative program integrated of Canary Islands 2014-2020 Strategy Aim 3, Priority Topic 74(85%); the Spanish network CAPAP-H, and the European COST Action CHIPSET.

The results presented in this paper are also partially funded from European Union's Horizon 2020 research and innovation programme under grant agreement No. 688201 (M2DC).

Referencias

[1] Hans Meuer, Erich Strohmaier, Jack Dongarra, and Horst Simon, "Top500 list, last accessed may 2019.," http://www.top500.org/.

[2] Innovative Computing Laboratory. Univ. Tennessee, "The parallel linear algebra for scalable multi-core architectures (PLASMA) project," `http://icl.cs.utk.edu/plasma/`, 2011.

[3] The FLAME Project, "Flame: Formal linear algebra methods environment," `http://z.cs.utexas.edu/wiki/flame.wiki/FrontPage`, 2011.

[4] Emmanuel Agullo, Jim Demmel, Jack Dongarra, Bilel Hadri, Jakub Kurzak, Julien Langou, Hatem Ltaief, Piotr Luszczek, and Stanimire Tomov, "Numerical linear algebra on emerging architectures: The PLASMA and MAGMA projects," *Journal of Physics: Conference Series*, vol. 180, no. 1, pp. 012037, 2009.

[5] Paul Richmond and Daniela Romano, "FLAME: Flexible large-scale agent modelling environment on the GPU," `http://www.flamegpu.com/`, 2010.

[6] Karan Singh, Major Bhadauria, and Sally A McKee, "Real time power estimation and thread scheduling via performance counters," *ACM SIGARCH Computer Architecture News*, vol. 37, no. 2, pp. 46–55, 2009.

[13] B. Rountree, D. H. Ahn, B. R. de Supinski, D. K. Lowenthal, and M. Schulz, "Beyond dvfs: A first look at performance under a hardware-enforced power bound," in *2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops PhD Forum*, May 2012, pp. 947–953.

[7] Gokcen Kestor, Roberto Gioiosa, Darren J. Kerbyson, and Adolfy Hoisie, "Enabling accurate power profiling of hpc applications on exascale systems," in *Proceedings of the 3rd International Workshop on Runtime and Operating Systems for Supercomputers*, New York, NY, USA, 2013, ROSS '13, pp. 4:1–4:8, ACM.

[8] Gregor Von Laszewski, Lizhe Wang, Andrew J Younge, and Xi He, "Power-aware scheduling of virtual machines in dvfs-enabled clusters," in *2009 IEEE International Conference on Cluster Computing and Workshops*. IEEE, 2009, pp. 1–10.

[9] Chia-Ming Wu, Ruay-Shiung Chang, and Hsin-Yu Chan, "A green energy-efficient scheduling algorithm using the dvfs technique for cloud datacenters," *Future Generation Computer Systems*, vol. 37, pp. 141–147, 2014.

[10] M. Curtis-Maury, A. Shah, F. Blagojevic, D. S. Nikolopoulos, B. R. de Supinski, and M. Schulz, "Prediction models for multi-dimensional power-performance optimization on many cores," in *2008 International Conference on Parallel Architectures and Compilation Techniques (PACT)*, Oct 2008, pp. 250–259.

[11] Xingfu Wu, Valerie Taylor, Jeanine Cook, and Philip J Mucci, "Using performance-power modeling to improve energy efficiency of hpc applications," *Computer*, vol. 49, no. 10, pp. 20–29, 2016.

[12] Charles Lively, Valerie Taylor, Xingfu Wu, Hung-Ching Chang, Chun-Yi Su, Kirk Cameron, Shirley Moore, and Dan Terpstra, "E-amom: an energy-aware modeling and optimization methodology for scientific applications," *Computer Science - Research and Development*, vol. 29, no. 3, pp. 197–210, Aug 2014.

[14] Aniruddha Marathe, Peter E. Bailey, David K. Lowenthal, Barry Rountree, Martin Schulz, and Bronis R. de Supinski, "A run-time system for power-constrained hpc applications," in *High Performance Computing*, Julian M. Kunkel and Thomas Ludwig, Eds., Cham, 2015, pp. 394–408, Springer International Publishing.

[15] Alberto Cabrera, Francisco Almeida, Javier Arteaga, and Vicente Blanco, "Measuring energy consumption using eml (energy measurement library)," *Computer Science - Research and Development*, vol. 30, no. 2, pp. 135–143, 2014.

[16] David Bailey, Tim Harris, William Saphir, Rob Van Der Wijngaart, Alex Woo, and Maurice Yarrow, "The nas parallel benchmarks 2.0," Tech. Rep., Technical Report NAS-95-020, NASA Ames Research Center, 1995.

PAS2P: Extendiendo análisis de aplicaciones paralelas e irregulares

Felipe Tirado [1] [2] Alvaro Wong [2] Dolores Rexachs [2] y Emilio Luque [2]

Resumen— **Cuando las herramientas analizan el rendimiento de aplicaciones con cientos de procesos, los datos generados pueden ser más grande que la memoria del nodo del cluster, causando que los datos sean cargados desde la memoria virtual. En HPC el uso de la memoria virtual no es una opción recomendada. El problema causa limitaciones en la escalabilidad y presenta un serio problema en la ejecución de programa a gran escala. Por otro lado existen aplicaciones científicas MPI que se caracterizan por tener un patrón de cómputo irregular en tiempo de ejecución, causando un problema al momento de analizar y predecir su comportamiento. Es por ello que el artículo aborda el problema del uso de memoria cuando las aplicaciones escalas a cientos de procesos, distribuyendo entre los nodos del cluster la generación de los datos de análisis, como también se propone una caracterización de aplicaciones MPI con patrón irregular de cómputo, basado en agrupamientos de procesos de acuerdo a los comunicadores MPI usados.**

Palabras clave— **Análisis rendimiento de aplicación, Aplicaciones paralelas MPI, Aplicaciones de cómputo irregular.**

I. Introducción

Durante los últimos años, los sistemas de cómputo de altas prestaciones, (High Performance Computing), han incrementado el número de unidades de procesamiento (CPU's) significativamente [1]. Actualmente, estos sistemas poseen una gran potencia de cómputo y todo parece indicar que esta tendencia continuará aumentando durante los próximos años, debido a las constantes mejoras tecnológicas, lo que hace posible el incremento tanto del número de cores por procesador, como del número total de procesadores en estos sistemas, estas mejoras han jugado un papel cada vez más importante en la investigación científica avanzada, ya que la simulación por computadora es utilizada ampliamente para aumentar la complejidad y el número de experimentaciones que se realizaban en forma manual.

Sin embargo, la brecha entre el rendimiento máximo y el rendimiento alcanzado en las aplicaciones científicas ejecutadas en paralelo ha aumentado considerablemente en los últimos años [2]. La compleja arquitectura de los sistemas paralelos, la interdependencia entre los diferentes componentes, junto con las estructuras de comunicación impuestas por el algoritmo, como también la existencia de modelos con un alto grado de irregularidad en su estructura, presentan retos difíciles a la hora de optimizar el rendimiento en aplicaciones paralelas científicas.

[1]Departamento de Computación e Industrias, Universidad Católica del Maule, Talca, Chili. E-mail: ftirado@ucm.cl

[2]Computer Architecture and Operating System Department, Universidad Autónoma de Barcelona, Barcelona, Spain. E-mail:{alvaro.wong, dolores.rexachs, emilio.luque }@uab.es

Los programa paralelos pueden presentar tres tipos de irregularidad [3]. La primera relacionada con las estructura de control irregular, que hacen ineficaz la ejecución en modelos de programación síncronas. Un segundo tipo aparece en estructuras de datos irregulares, como son: árboles desbalanceados, grafos y datos no estructurados. El tercer tipo de irregularidad son los patrones de comunicación irregulares, que conducen al no-determinismo, ya que no se puede predecir el orden en que ocurrirán los eventos de comunicación.

Existen numerosas herramientas que recopilan y presentan información relevante sobre el rendimiento de las aplicaciones a un alto nivel de abstracción para que los desarrolladores puedan identificar y determinar fácilmente las causas que afectan al rendimiento en las aplicaciones.

En el proceso de instrumentación de una aplicación, por lo general se realiza a través de seguimiento de evento, que es un método para analizar el comportamiento de aplicaciones paralelas. Los eventos se registran a lo largo de la ejecución de la aplicación, los cuales representan acciones relevantes para el análisis de la aplicación como son, él envió y recepción de mensajes o la ejecución de alguna rutina de importancia [2], para luego ser analizados con la ayuda de algunas herramientas de software como puede ser VAMPIR [4] o Paraver [5] , los cuales permite visualizar el comportamiento de aplicaciones paralelas mediante la instrumentación de los eventos, como también proveer resumen estadístico del comportamiento de la comunicación.

La instrumentación de una aplicación ejecutada en miles de procesos, acarrea un serio problema de tamaño de dato, el cual se incrementa considerablemente al ir escalando la aplicación, debido al gran volumen de eventos que se generan, llegando a alcanzar un tamaño mayor al de la memoria del nodo computacional, causando una limitación de escalabilidad en la aplicación. Este problema de memoria hace que la programación distribuida sea necesaria para dividir los datos entre todos los nodos del cluster, evitando una baja en el rendimiento producto del uso de la memoria virtual.

La herramienta PAS2P (Parallel Application Signature for Performance Prediction) [6] [7] propone la extracción de la firma de aplicaciones paralela de paso de mensajes (MPI) . Esta firma, representa el comportamiento significativo de la aplicación, ya que está construida seleccionando únicamente las partes relevantes (fases) del código de la aplicación, es decir, las que tiene impacto en el rendimiento, permitiendo caracterizar la aplicación bajo una máquina de des-

tino. Una fase esta definida como un segmento de código paralelo delimitados por comunicaciones MPI que se repite (peso) a lo largo de la ejecución. La herramienta PAS2P consiste en dos etapas. La primera etapa es la instrumentación y el análisis de la aplicación para obtener las fases y la generación de la firma. La segunda etapa consiste en la ejecución de la firma para predecir el tiempo de ejecución de la aplicación.

El analizador PAS2P, necesita procesar la información para todo los procesos encontrado las dependencias, con el fin de aplicar la ordenación lógica entre ellos y construir un reloj lógico global que mantenga la precedencia entre los eventos de comunicación. Cuando la aplicación es ejecutada con un elevado número de procesos se pueden encontrar que no existe suficiente memoria en el nodo, por lo que el uso memoria virtual es necesario, incrementando considerablemente el tiempo de ejecución o en muchos casos ni siquiera es posible su ejecución.

Además de lo anterior, existen aplicación científicas MPI con un alto grado de irregularidad, lo que produce un comportamiento de cómputo no predecible, dificultando la caracterización de la aplicación al momento de general la firma, debido principalmente al comportamiento dinámico de la aplicación que dificulta la reducción de las fases.

En la siguiente sección del artículo, se presenta los trabajos relacionados. En la Sección 3 presentamos una descripción de PAS2P y en la Sección 4 se presenta una visión general de las aplicaciones de comportamiento irregular. La Sección 5 se provee la paralelización del módulo analizado, la Sección 6 muestra una caracterización de las aplicaciones con procesos irregulares, posteriormente en la Sección 7 se presentan los resultados de la paralelización del módulo analizador, por último en la Sección 8 se exponen las conclusiones y trabajos futuros.

II. Trabajo Relacionado

Existen otras herramientas de análisis que también poseen limitaciones de escalabilidad que afectan negativamente la experiencia del usuario cuando la herramienta se utiliza a gran escala. Scalasca [8] es un conjunto de herramientas de código abierto que puede ser utilizado para analizar el comportamiento de rendimiento de aplicaciones paralelas e identificar puntos de optimización. Aunque Scalasca puede ser utilizado sobre 294,912 núcleos, la versión 1.3.0 y las versiones subyacentes mostraron limitaciones de escalabilidad durante la recopilación y visualización de los informes de análisis. Los desarrolladores se centraron en mejorar estas carencias.

TAU [9] y Vampir [10] han centrado sus esfuerzos en mejorar el análisis de datos a gran escala. Para lograr este análisis, TAU utiliza ParaProf (Parallel Profile Analyzer) [11], que fue construido específicamente para el análisis de datos a gran escala. El análisis se realiza en memoria para un acceso rápido y un soporte global para las vistas de análisis. TAU proporciona un formato de datos comprimido y normalizado, como contenedor de datos para cualquier herramienta de medición soportada. Esto hace que la lectura en paralelo de los profiles sea significativamente más eficiente en ParaProf. Vampir proporciona una eficiente acceso a la traza de archivo. Esto permite distribuir los datos en varios ficheros, cada uno de los cuales almacena un "frame" de los datos de ejecución. Los frames pueden corresponder a una sola CPU o a un grupo de CPUs. Los frames pertenecientes a una misma ejecución hacen uso de índice de archivo para proporcionar un mejor rendimiento.

F. Wolf et al [2] a través de trazas de eventos registran las relaciones temporales y espaciales en tiempo de ejecución, permitiendo analizar dependencias de rendimiento en los flujos de control concurrentes, sin embargo, debido a la gran cantidad de datos generados en las máquinas paralelas, la amplitud y profundidad del análisis realizado es a menudo limitada, es por ello que el autor automatiza la búsqueda de patrones de comportamiento ineficiente en los eventos registrados, clasificándolos por categoría y cuantificando su penalización por rendimiento asociado. Esto permite que los desarrolladores estudien el rendimiento de sus aplicaciones en mucho menor tiempo y con una menor experiencia que un análisis manual. Dentro de los trabajos futuros de esta investigación, hace mención a que su propuesta no presenta una buena escalabilidad, debido al gran tamaño de su archivo de traza de eventos. Existen trabajos [7] que son capaces de predecir el comportamiento de escalabilidad de las aplicaciones de paso de mensajes en sistemas específicos. La metodología propuesta se centra en la caracterización y análisis de los patrones de comunicación, a partir de un conjunto de ejecuciones en pequeña escala, para proyectar su comportamiento cuando el número de procesos aumenta. Este modelo permite predecir el tiempo de calculo con alta precisión para una gran número de procesos, utilizando la metodología de análisis de comportamiento a través de firmas [7].

A. Morari et al [12] menciona que el software de próxima generación tendrá que explotar más modelos de programación para usar adecuadamente las nuevas generaciones de los sistemas HPC, a la ves de mantener un bajo overhead. Las tendencias actuales muestran que muchas de las funcionalidades del sistema operativo tradicional, como la planificación de tareas, balanceo de carga, mecanismos de interconexión, entro otros, se tornarán cada vez mas complejo su realización en tiempo de ejecución, como también permitir ejecutar diversas aplicaciones con diferentes modelos de programación. Una clasificación típica distingue entre aplicaciones regulares e irregulares. Las aplicaciones regulares poseen un patrón de comunicación y acceso a datos bastante regular y predecible, en cambio las aplicaciones irregulares tienen un patrón de comunicación y acceso de memoria muy irregular, por lo tanto, una muy mala ubicación espacial y temporal. Esta investigación propone un modelo que reduce los incon-

venientes de trabajar con este tipo de aplicaciones en sistema HPC, cuantificando la degradación del rendimiento.

III. Descripción de PAS2P

La herramienta PAS2P esta basada en la repetitividad de la aplicación paralela, centrada en el análisis y predicción del rendimiento de la aplicación MPI utilizando su firma. La herramienta PAS2P, como se muestra en la Fig. 1, se divide en dos etapas, una primera etapa de generación de la firma y una segunda etapa de ejecución de la firma.

La primera etapa está compuesta por tres módulos. El primer modulo (modulo de instrumentación) instrumenta cada proceso de la aplicación creando un archivo de traza por proceso. Los archivos de datos contiene las llamadas a las primitivas MPI con sus respectivos contadores de hardware. La instrumentación es realizada por la librería dinámica PAS2P junto con la integración de la librería PAPI [13] para los contadores de hardware.

Una vez obtenidas las trazas de forma secuencial por el módulo de instrumentación, se continua con el segundo módulo (módulo analizador), el cual tiene como objetivo crear un modelo de aplicación independiente de la máquina. Para ello es necesario crear un reloj lógico global para todo los procesos con el fin de mantener la precedencia entre los eventos, se define como evento la acción de recibir o enviar un mensaje MPI. El algoritmo está inspirado por el método de Lamport [14] en el que, si un proceso envía un mensaje en un tiempo lógica (LT) su recepción llegará en un tiempo $LT + 1$.

Una vez ordenado todos los eventos, se crea el trazado lógico, donde se insertan los eventos para luego ser analizados y extraer las fases de la aplicación. Una vez que tenemos las fases, se almacena la información en el archivo *Phase_Table*, que será la salida del módulo analizador. Para construir la firma de la aplicación cada proceso del Módulo Generador de Firma leerá el archivo *Phase_Table*. La segunda etapa ejecuta el último módulo (Módulo de Predicción de Rendimiento), ejecutado en paralelo y usado para predecir el rendimiento de la aplicación en una máquina de destino.

A diferencia de los otros módulos, el Módulo Analizador observado en la Fig. 2, es un programa en serie que analiza todo la traza generada por la instrumentación, tomando un tiempo considerable en extraer las fases. En algunos casos, dependiendo del tamaño de los archivos de traza, que viene dado por el número de procesos y los eventos almacenado, la memoria requerida para cargar los datos en memoria principal puede ser de mayor tamaño que la memoria del nodo, forzando al sistema el uso de memoria virtual. Por esta razón, hemos paralelizado el Módulo Analizador con el fin de analizar las trazas en forma paralela, usando el mismo número de recursos que la aplicación utiliza para su ejecución. Esto asegura la reducción de el tiempo de análisis y la distribución del uso de memoria, proporcionando al usuario una

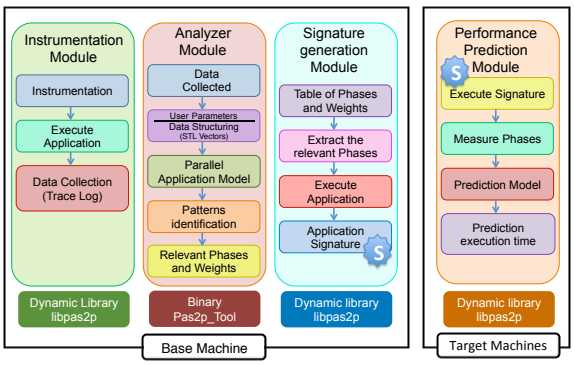


Fig. 1. Etapas de la metodología PAS2P.

herramienta que puede ser utilizada para analizar y predecir el rendimiento de aplicaciones a gran escala.

IV. Aplicaciones de Comportamiento Irregular

El comportamiento irregular ocurre comúnmente en diversas aplicaciones científicas. Aunque son inherentemente paralelas, exhiben un rendimiento de ejecución altamente variable debido a que presentan patrones de acceso a la memoria y/o patrones de comunicación indeterministas, estructura de basadas en punteros y no exhiben un balanceo de carga adecuado al sistema en donde se ejecuta. Además de requerir de una sincronización y comunicación de grano fino en estructuras de datos grandes, como grafos, arboles desbalanceados, grillas no estructuradas. Debido al comportamiento no deterministico del patrón de acceso a los datos, estas aplicaciones tienen una localización espacial y temporal muy pobre, pero a pesar de eso posee un alto grado de paralelismo, que es difícil de explotar debido a su comportamiento complejo. Además, las aplicaciones irregulares son difíciles de escalar en las máquinas de supercomputación actuales, debido a sus límites en la sincronización de grano fino y las pequeñas transferencias de datos.

Las aplicaciones irregulares tienen un campo bien establecido como el aprendizaje automático, el análisis de redes sociales, la bioinformática, las bases de datos de gráficos semánticos, el diseño asistido por computadora (CAD) y la seguridad informática. Muchas de estas áreas de aplicación también procesan conjuntos masivos de datos no estructurados, que siguen creciendo de manera exponencial.

V. Paralelización del Módulo Analizador

Con el objetivo de evitar los accesos a la memoria virtual y reducir el tiempo del análisis de la aplicación, se ha paralelizado el módulo de análisis mediante el paradigma de memoria distribuida observado en la Fig. 3, utilizando la librería de paso de mensajes MPI, con la finalidad de aprovechar los recursos utilizados por el módulo de instrumentación para ejecutar la aplicación.

El módulo analizador paralelo, tal y como se muestra en la Fig. 4, se compone de 3 etapas: la cargar datos, donde cada proceso del módulo analizador lee

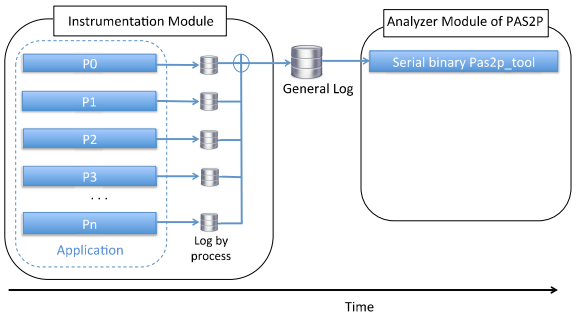


Fig. 2. Módulo Analizador Serie.

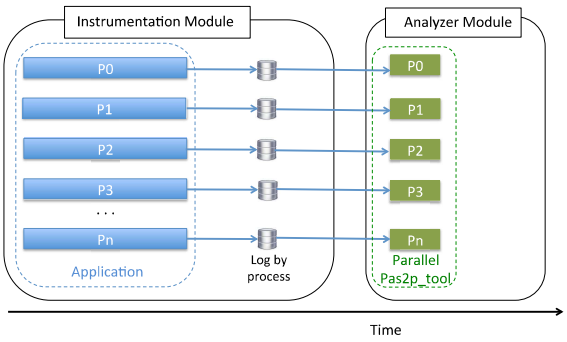


Fig. 3. Módulo Analizador Paralelo.

su archivo de traza de datos. Luego, se insertan el Tiempo Lógico a los eventos en el etapa del modelo de la aplicación. Por último en la etapa de identificación del patrón, se utilizan la comunicación colectiva, debido a que existen dependencias de datos que se necesitan sincronizar. En esta sección explicaremos las mejoras realizadas en el módulo analizador y cómo se paralelizo la implementación.

A. Carga de datos

Con el objetivo de generar el modelo abstracto de la aplicación, el módulo serie concatena los logs de trazas de cada proceso en un log global, el cual será cargado en memoria con el objetivo de ser analizado en serie.

A diferencia del módulo serie, el módulo paralelo aprovecha los recursos utilizados durante la ejecución de la aplicación, creando el mismo número de procesos que fueron utilizados para ejecutar la aplicación, tal y como se muestra en la figura 3. Una vez que los procesos han sido creados, cada proceso carga su propio log de traza, de esta forma se consigue reducir la memoria total por nodo.

B. Modelo abstracto de la aplicación

Una vez se ha cargado el log de traza, el siguiente paso es crear el modelo abstracto de la aplicación, para ello, es necesario dotar a cada evento de comunicación de un Tiempo Lógico (TL), mediante el cual ordenar los eventos de comunicación. Con el objetivo de ordenar estos eventos, se utiliza un algoritmo basado en Lamport [14].

Puesto que cada proceso tiene su log de traza, es necesario enviar el Tiempo Lógico (TL) entre los diferentes procesos, con el objetivo de asignar a los eventos de recepción su TL. Tomando ventaja de la distribución de los datos, a la hora de diseñar el algoritmo, se optó por replicar el patrón de comunicación de la aplicación para enviar el TL entre los diferentes procesos. Para ello, si el algoritmo encuentra una primitiva de envío la convierte en un *MPI_Send*, y envía su TL como contenido del mensaje, mientras que si es un mensaje de recepción se convierte en una primitiva *MPI_Recv*. Cuando una colectiva es detectada, se convierte en una primitiva *MPI_Allreduce*, y se selecciona el máximo TL de todos los procesos.

El orden del algoritmo paralelo es mostrado en la Fig. 5, cada proceso realiza los siguientes pasos.

1. Una cola para cada proceso es creada y el primer evento de cada proceso es insertado en cada cola.
2. El primer evento (CurrentEvent) es sacado de la cola de cada proceso.
3. El siguiente evento a CurrentEvent (ForwardEvent) es insertado en la cola de cada proceso.
4. Se busca el proceso anterior (BackEvent) al currentEvent con el TL mayor y un tiempo físico menor que CurrentEvent. Si hay varios eventos con el mismo TL y uno de ellos es de emisión, se toma este. Si no, se toma cualquier evento de recepción. Si el BackEvent es una emisión, el Tiempo Lógico (TL) del CurrentEvent se incrementa un unidad el Tiempo Lógico con respecto al BackEvent, por último, si BackEvent es una recepción, el Tiempo Lógico de currentEvent será igual al Tiempo Lógico del BackEvent más uno.
5. Si el evento CurrentEvent es un envío, se crea un mensaje *MPI_Send* el cual tiene como dato el TL de CurrentEvent, y se usa el ID Relación como MPI_TAG del mensaje. Si el CurrentEvent es un evento de recepción se crea *MPI_Recv* que recibe el Tiempo Lógico del CurrentEvent y el ID Relation como el MPI_TAG del mensaje y se asigna el Tiempo Lógico del RecvEvent como el incremento en una unidad del Tiempo Lógico del CurrentEvent (RecvEvent = CurrentEvent+1).
6. El proceso es completado cuando la cola quede completamente vacía.

Una vez que todos los eventos tienen asignado un tiempo lógico, se crea una traza lógica con el modelo abstracto de la aplicación. A diferencia de la traza lógica en serie, la traza lógica generada por el módulo paralelo está distribuida entre los diferentes procesos de la aplicación, teniendo únicamente una sola dimensión, la longitud máxima es el número máximo de eventos (ticks) de la aplicación. De esta manera conseguimos reducir el espacio de memoria del nodo de cómputo.

C. Identificación de patrones

El objetivo de esta etapa es identificar las fases de la aplicación paralela, PAS2P extrae las fases directamente desde la traza lógica. En la versión serie del

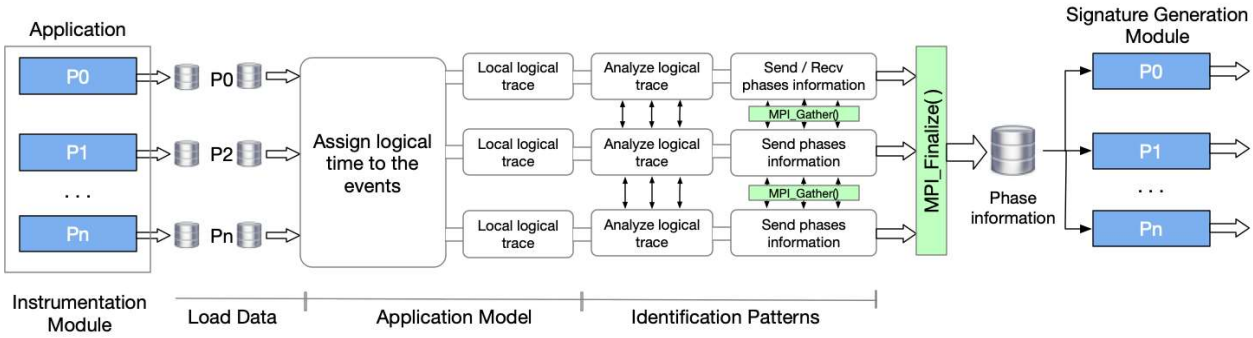


Fig. 4. Visión general de la paralelización del módulo analizador.

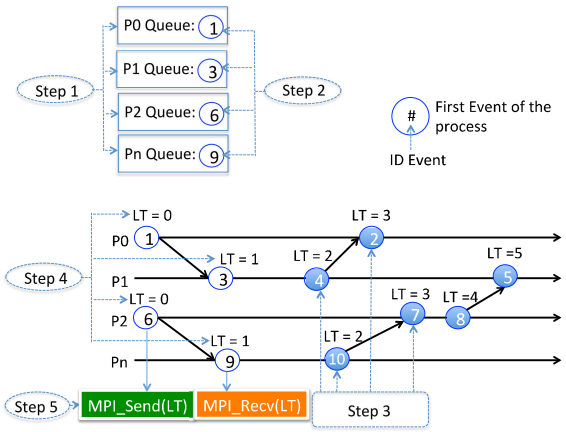


Fig. 5. Insertar tiempos lógicos utilizando el algoritmo de ordenación paralelo.

analizador, la traza es cargada en la memoria de un nodo, permitiendo tener una visión global de la traza lógica, lo que permite buscar la repetitividad de los eventos en todos los procesos de la aplicación, mientras en la versión paralela del Módulo Analizador, cada proceso tiene una traza lógica local, como se muestra en la Fig. 4. Esta distribución de traza presenta el inconveniente de que no se conoce en que tick y proceso se produce un nuevo evento.

Como dijimos anteriormente, cada proceso del analizador paralelo crea su traza lógica local, lo que significa que cada proceso tiene sus eventos. Teniendo en cuenta la distribución de datos, tenemos que sincronizar los procesos con el fin de poder comparar los eventos de un mismo Tiempo Lógico (tick). Manteniendo la sincronización, cada proceso tiene que buscar la similitud local en la traza lógica local y luego todos los procesos de PAS2P se comunican para comparar si existe una fase (similitud global).

Como se muestra en la Fig. 6, para sincronizar los procesos se crea una llamada colectiva *(MPI_Allreduce)* entre cada tick, de esta manera todos los procesos saben si un evento se repite. Observando nuevamente la Fig. 6, cuando se detecta un evento repetido, cada proceso cuenta los eventos similares (similitud local) de la nueva fase con la fase existente. En esta parte todo los procesos tienen su contador local de eventos similares. Para conocer la similaridad de la fase se crea otra comunicación colectiva *(MPI_Allreduce)* para añadir todo los contadores locales de los eventos similares (similitud global). Cuando todos los procesos tienen un contador global similar, se procede a utilizar le algoritmo serial de similaridad de fase definido por los siguientes criterios.

1. El tamaño de la fase (número de ticks) a comparar debe ser el mismo.
2. Para comparar los eventos de una fase, se aplican los siguientes criterios:
   - Se compara si dos eventos tienen el mismo tipo de comunicación y similar volumen de comunicación (5% de diferencia).
   - El tiempo de cómputo entre dos eventos, si dos eventos tienen tiempos similares (mayor o igual a 85%)
3. Una fase es similar, si el número de eventos similares es mayor o igual que 80% (valor configurable) del número total de eventos que componen la fase.
   - Si una fase es similar, el peso de la fase existente se incrementa. Si no es similar, es almacenada como una fase nueva.

Una vez creadas las fases, cada proceso tiene la información local sobre sus fases. Se define como proceso maestro al proceso con identificador 0, el cual necesita la información almacenada en todos los procesos restantes del analizador. Para compartir estos datos con el proceso principal, se utiliza un mecanismo MPI basado en grupos (*MPI_Gathers*). Como se observa en la Fig. 7 y Fig. 8, donde n es el Rank del proceso MPI y m es el identificador del nodo.

Este método de reducción permite minimizar la cantidad de comunicación en la red. Una vez que la información es obtenida por el proceso maestro,

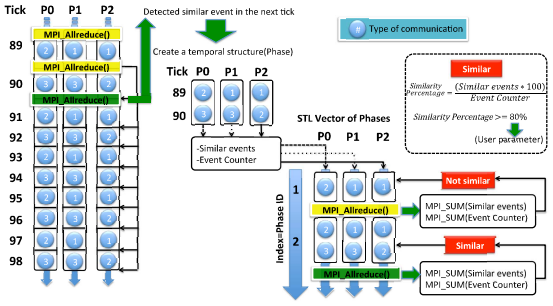


Fig. 6. Paralización del algoritmo de identificación de fases.

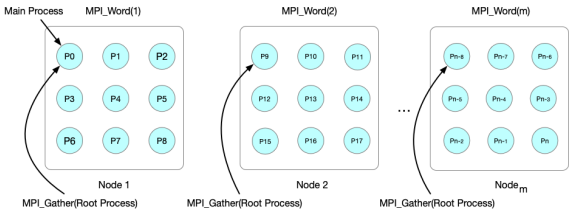


Fig. 7. Solicitud datos por el proceso principal de cada grupo.

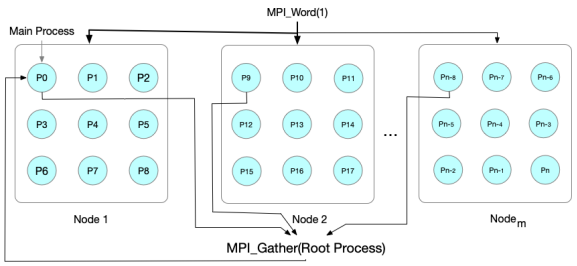


Fig. 8. Solicitud de datos del proceso maestro a los procesos principales de cada grupo.

este muestra la información sobre las fases y su rendimiento.

VI. Caracterización de aplicaciones con procesos irregulares

Una vez paralelizado el Módulo Analizador de PAS2P, se extiende la capacidad de análisis a otro modelo de programación paralela caracterizado por aplicaciones que presentan comportamiento de cómputo irregular [3] entre sus procesos, este comportamiento se debe principalmente a la estructura de control irregular que presenta la aplicación.

La anterior situación se puede ver ilustrada en la Fig. 9, donde se observa que al ejecutar una aplicación MPI, existen grupos de procesos que realizan un tipo de cómputo y otros grupos de procesos realizan otro tipo de cómputo, el mundo 0 realiza operaciones matriciales, el mundo 1 realiza operaciones vectoriales y el mundo 2 realiza operación basadas en árboles, es decir, tenemos una misma aplicación MPI, compuesta por 10 procesos agrupados de cierta manera que permiten realizan

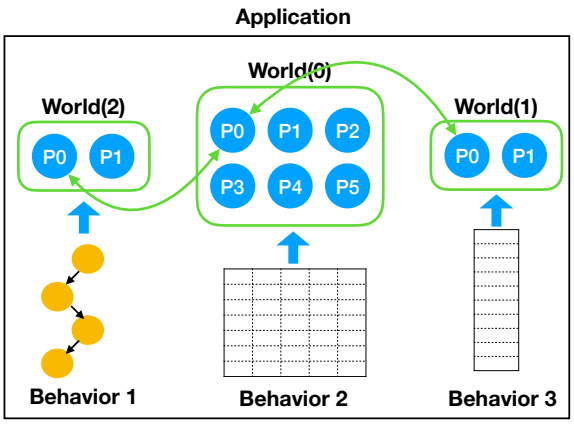


Fig. 9. Aplicación comportamiento irregular.

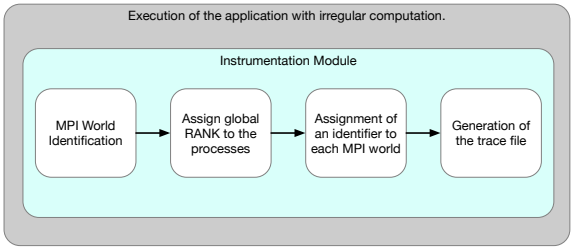


Fig. 10. Módulo instrumentador de aplicaciones con comportamiento irregular.

cómputo distinto. Este comportamiento por lo general se aprecia al procesar estructuras de datos irregulares o simplemente porque la aplicación se desarrolló agrupando procesos que realicen distinto tipo de cómputo.

La herramienta PAS2P se basa en el comportamiento repetitivo de la aplicación, obteniendo una representación basada en firmas que caracteriza la aplicación MPI. El problema del modelo de programación de cómputo irregular, es que no existe un patrón común de repetitividad debido a que los proceso o grupos de proceso realizan diverso cómputo, lo que dificulta el análisis de PAS2P, o más aun, siendo imposible elaborar una firma para la aplicación.

Las aplicaciones MPI de cómputo irregular, logran esta característica realizando grupos de procesos que poseen su propio comportamiento de cómputo y patrón de comunicación. Por lo general esta característica esta relacionada con la creación de comunicadores que agrupan los procesos bajo un MPI_WORD. Un ejemplo de esto es el uso de las primitivas MPI para la creación de comunicadores con topología cartesiana como son: *MPI_Card_create, MPI_Card_sub y MPI_Card_shift.*

El instrumentador de PAS2P, como se mencionó anteriormente, es el encargado de producir una traza de la aplicación que posteriormente se utiliza para la recolección de datos, con fin de caracterizar el comportamiento del cómputo y las comunicaciones. Para lograr esto, en las aplicaciones con irregularidad de cómputo, es necesario agregar información adicional a la traza de la aplicación que permita caracterizar y general un modelo. Esta información se obtendrá interceptando las llamadas MPI de creación de nuevos comunicadores (MPI WORD). Cuando el analizador de PAS2P intercepta las llamadas, realiza una identificación global basada en el *MPI_Comm_rank* que posee cada proceso en el comunicador principal MPI_COMM_WORLD. Una vez que cada proceso tiene un identificador único dentro de la aplicación, se procede a etiquetar los diversos comunicadores que representan a cada mundo con un número entero único. Hasta este punto tenemos identificadas diversas instrucciones MPI, las cuales son asociadas al identificador del mundo en la que se realiza su llamada, para luego generar el archivo de traza de la aplicación. Este procedimiento descrito anteriormente se observa en el Fig. 10.

La generación de la traza de la aplicación MPI es

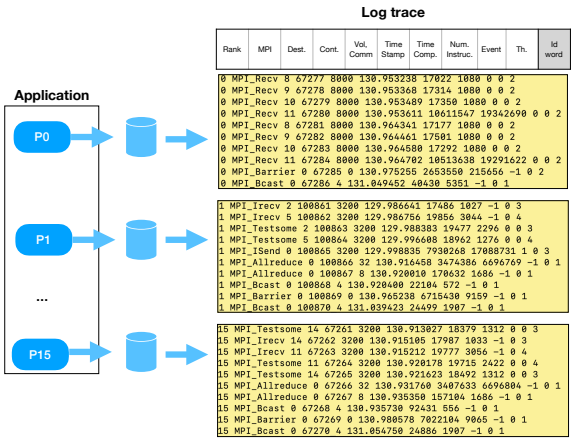


Fig. 11. Archivo de traza identificado mundos MPI.

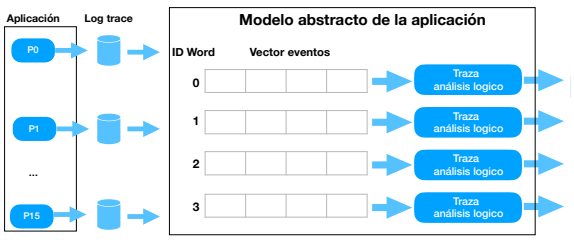


Fig. 12. Modelo aplicación agrupados por comunicadores.

almacenada por proceso en archivos logs, como se explico en la sección de paralelización del Módulo Analizador, cada proceso entrega información relevante con respecto a su comportamiento, tal como se muestra en la Fig. 11. Como se puede observar en la Fig. 11, la traza proporciona datos como: origen del mensaje MPI, tipo de primitiva utilizada, destino del mensaje MPI, contador de instrucciones, volumen de comunicación en byte, el tiempo de cómputo entre eventos en nanosegundos, número de instrucciones, el tipo de eventos y en la última columna el identificador del mundo que realiza el llamado a la función MPI.

Esta información adicional agregada en la traza, nos permitirá diseñar un modelo de la aplicación, agrupando los diversos eventos por el identificador atribuido a cada mundo MPI, en este modelo se asignan los tiempos lógicos a los eventos provenientes de cada comunicador para posteriormente realizar la traza lógica local de cada vector de eventos, como se observa en la Fig. 12

La obtención de un modelo de aplicación basado en comunicadores, nos permitirá caracterizar la aplicación de acuerdo al comportamiento de los mundos MPI, extrayendo las fases de cada uno de ellos, con el fin de construir una firma representativa de la aplicación de computo irregular.

VII. Resultados de la Paralelización del Módulo Analizador

En esta sección se presenta el tiempo requerido por el analizador paralelo para obtener los resultados. La experimentación consiste en la ejecución de una aplicación con diversas configuraciones en una máquina

TABLA I
Especificaciones Cluster.

Cluster	Characteristics
DELL	AMD Opteron™ 6200 1.60GHz, 8 nodes (512 cores), 64 GB RAM per node, Interconnection Gigabit Ethernet.

TABLA II
Rendimiento serial y paralelo en el Cluster Dell.

Cluster	Application	Number Processes	Data Size (GB)	Serial Approach Time (sec)	Parallel Approach Time (sec)
Dell	LU	32	0.378	189.66	9.71
		64	0.805	751.00	15.11
		128	1.700	2281.89	173.80
		256	3.500	8928.84	319.29
		512	7.200	41084.40	469.48

destino, para validar como la implementación paralela mejora el tiempo de análisis.

Para la ejecución de las aplicaciones se uso una máquina de destino caracterizada en la Tabla I. Para los experimentos se uso la aplicación LU de la NPB [15] compilada para 32 a 512 procesos, usando la clase C como carga de trabajo.

Para el cluster DELL, se ejecuto la aplicación LU con la instrumentación de PAS2P para general los archivos de datos. Estos fueron generados con la versión serial y la versión paralela del analizador para contrastar los tiempos requeridos para analizar los datos de la aplicación. Cuando los procesos se incrementan el tamaño del archivo de datos también se incrementa, como se observa en la Tabla II, ya que aumenta el numero de mensajes MPI, es decir, el tamaño del archivo de datos es dependiente del patrón de comunicación de la aplicación.

En la Tabla II, se observa que existe un aumento considerable en el tiempo de ejecución paralela después de los 64 procesos, esto se debe a que el Cluster DELL contiene 64 núcleos por nodo, al utilizar una mayor cantidad de proceso, la versión paralela de PAS2P comienza a utilizar la red de interconexión.

VIII. Conclusiones y Trabajo Futuro

En este artículo presentamos una extensión del módulo de análisis de la herramienta PAS2P que resuelve las limitaciones de escalabilidad con respecto a la extracción del comportamiento de una aplicación MPI a gran escala. Para eso se diseño una paralelización del algoritmo del Módulo analizador, con el fin de eliminar el problema de memoria y reducir el tiempo del análisis de la aplicación, tomando ventaja de todos los recursos usados para ejecutar la aplicación.

Además proponemos una modelo de caracterización para aplicaciones con cómputo irregular, basado en la agrupación de procesos por comunicadores, con el fin de generar un modelo de la aplicación paralela extrayendo las fases más significativas, creando una firma que permita analizar y predecir el rendimiento de la aplicación en diferentes máquinas paralelas.

Como trabajo futuro se seguirá nuestro trabajo de extender la herramienta PAS2P hacia el análisis de

aplicaciones de cómputo irregular, elaborando una firma que permita caracterizar el comportamiento de la aplicación.

Agradecimientos

This research has been supported by the Agencia Estatal de Investigación (AEI), Spain and the Fondo Europeo de Desarrollo Regional (FEDER) UE, under contract TIN2017-84875-P and partially funded by a research collaboration agreement with the Fundacion Escuelas Universitarias Gimbernat (EUG).

Referencias

[1] Norbert Attig, Paul Gibbon, and Th Lippert, "Trends in supercomputing: The european path to exascale," *Computer Physics Communications*, vol. 182, no. 9, pp. 2041–2046, 2011.

[2] Felix Wolf, Bernd Mohr, Jack Dongarra, and Shirley Moore, "Automatic analysis of inefficiency patterns in parallel applications," *Concurrency and Computation: Practice and Experience*, vol. 19, no. 11, pp. 1481–1496, 2007.

[3] Katherine A Yelick, "Programming models for irregular applications," *ACM SIGPLAN Notices*, vol. 28, no. 1, pp. 28–31, 1993.

[4] Wolfgang E Nagel, Alfred Arnold, Michael Weber, Hans-Christian Hoppe, and Karl Solchenbach, "Vampir: Visualization and analysis of mpi resources," 1996.

[5] Jesús Labarta, Sergi Girona, Vincent Pillet, Toni Cortes, and Jose Maria Cela, "A parallel program development environment," in *In proceedings of the 2th. Int. Euro-Par Conference*. Citeseer, 1995.

[6] Javier Panadero, Alvaro Wong, Dolores Rexachs, and Emilio Luque, "A tool for selecting the right target machine for parallel scientific applications," in *ICCS*, 2013, pp. 1824–1833.

[7] Alvaro Wong, Dolores Rexachs, and Emilio Luque, "Parallel application signature for performance analysis and prediction," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 7, pp. 2009–2019, 2015.

[8] Markus Geimer, Pavel Saviankou, Alexandre Strube, Zoltán Szebenyi, Felix Wolf, and Brian J. N. Wylie, "Further improving the scalability of the Scalasca toolset," in *Proc. of PARA 2010: State of the Art in Scientific and Parallel Computing, Part II: Minisymposium Scalable tools for High Performance Computing*, 2012, vol. 7134 of *Lecture Notes in Computer Science*, pp. 463–474.

[9] Sameer Shende, A Malony, G Allen, J Carver, Sct Choi, T Crick, and MR Crusoe, "Using tau for performance evaluation of scientific software," in *Workshop on Sustainable Software for Science: Practice and Experiences*, 2016, number 1686.

[10] Holger Brunst, Hans-Christian Hoppe, Wolfgang E. Nagel, and Manuela Winkler, "Performance optimization for large scale computing: The scalable vampir approach," in *ICCS*, 2001, pp. 751–760.

[11] Sameer Suresh Shende, Allen D. Malony, and Alan Morris, "Improving the scalability of performance evaluation tools," in *Proc. of PARA 2010*, 2010, pp. 441–451.

[12] Alessandro Morari and Mateo Valero, "Hpc system software for regular and irregular parallel applications," in *2013 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum*. IEEE, 2013, pp. 2242–2245.

[13] Dan Terpstra, Heike Jagode, Haihang You, and Jack Dongarra, "Collecting performance data with papi-c," in *Tools for High Performance Computing 2009*, pp. 157–173. Springer, 2010.

[14] Leslie Lamport, "The ordering of events in a distributed system," *Commun. ACM*, vol. 21, no. 7, pp. 558–565, July 1978.

[15] David H. Bailey, E. Barszcz, J. T. Barton, and Browning D. S., "The nas parallel benchmarks," in *Proceedings of the 1991 ACM/IEEE Conference on Supercomputing*, 1991, Supercomputing '91, pp. 158–165.

Lenguajes, compiladores y herramientas de programación y ejecución paralela

Tasks Fairness Scheduler for GPU

B. López-Albelda, J.M. González-Linares y N. Guil[1]

Abstract— **Nowadays GPU clusters are available in almost every data processing center. Their GPUs are typically shared by different applications that might have different processing needs and/or different levels of priority. As current GPUs do not support hardware-based preemption mechanisms, it is not possible to ensure the required Quality of Service (QoS) when application kernels are offloaded to devices. In this work, we present an efficient software preemption mechanism with low overhead that evicts and relaunches GPU kernels to provide support to different preemptive scheduling policies. We also propose a new fairness-based scheduler named Fair and Responsive Scheduler, (FRS), that takes into account the current value of the kernels slowdown to both select the new kernel to be launched and establish the time interval it is going to run (quantum). Nine applications selected from different benchmark suites have been used to evaluate the computing cost and the eviction delay of our preemption mechanism, showing our implementation has a very low overhead. The proposed scheduler has also been compared with Shortest Job First Scheduler (SJF), Shortest Remaining Time Scheduler (SRT), Round Robin Scheduler (RR) and Completely Fairness Scheduler (CFS). Comparison was carried out using different metrics like average normalized turnaround time (ANTT), deviation normalized turnaround time (DNTT) or system overall throughput (STP). DNTT metric shows that FRS consistently obtains better fairness scheduling results than other schedulers, specifically FRS is 1.5 times lower (better) than SRT (the second best fair scheduler).**

Keywords— **GPU preemption, Scheduling, CUDA Streams.**

I. Introduction

GPUs are broadly used in multitask environments, such as data centers, where applications running on CPUs offload specific functions to GPUs in order to take advantage of the device high performance. In these environments, it is likely to have several independent kernels ready to run concurrently on a GPU.

In this context, several works have been published that try to improve the way kernels are scheduled on GPUs. They pursue different aims like reducing the makespan of a set of kernels by taking advantage of concurrent kernel execution capabilities available in devices [1], [2], or providing priority-based kernel execution by developing soft-real time schedulers [3]. Implementing more complex scheduling policies which provide quality of service (QoS), fairness, and support for different priorities requires the ability to preempt running kernels, i.e., evict a kernel before finishing execution and schedule a new kernel. Unfortunately, although NVIDIA GPUs support some sort of compute preemption since Pascal architecture [4], it is restricted to a few uses such as interactive graphics tasks and debuggers, and it is not exposed to programmers. There are some research proposals of hardware-based preemption mechanisms, [5], [6], but they are not available in real GPUs, and there is no guarantee that they would be more efficient than software-based approaches. The implementation of a preemption mechanism on GPUs, similar to those employed on CPU, would require to save the state of all active threads (e.g., the contents of the entire register file) in the running kernel. As thousands of threads can be active in a standard GPU, the cost of collecting and saving all these states would be very high and incompatible with the implementation of an efficient preemptive scheduler, where kernels should be evicted and relaunched with minimum overhead.

More recently, other works have proposed the use of software-based preemption mechanisms to improve the scheduling policies [7], [8]. Thus, these methods only need to keep track of the number of pending thread blocks. As a result, they avoid the execution of expensive operations to save kernel state. Our preemptive scheme also follows this thread block-based approach. The mechanisms proposed by [7], [8] are based on zero-copy memory transactions between GPU and CPU memories. In addition, they require to run a proxy kernel on GPU to reduce the PCI data traffic generated during preemption. Our scheduler will not use a GPU proxy. Therefore, our scheduler sends scheduling commands to the running kernel by waiting GPU global memory variables. This can reduce the traffic through the interconnecting bus and use all the GPU resources only for kernel execution.

Regarding scheduling policies, previous works have implemented systems based on simple priority, priority queues or round robin schemes, [3], [9], [8] among others. However, not many works have studied fairness based policies to build GPU schedulers. We think fairness-oriented policies are necessary in GPUs servers where users demand a fair distribution of the available resources.

With the aim of improving current techniques for GPU kernel scheduling, we use explicit transfers to deploy our preemption mechanism We also develop a Fair and Responsive scheduler, FRS, that ensures, in a holistic way, fairness across different GPU applications. Thus, we make two main contributions:

- An efficient preemption mechanism that directly sends scheduling commands to the running kernel using GPU global memory variables. This way, we avoid the use of a proxy kernel, which allows application kernels to have more GPU resources available and, at the same time, eliminates the constant PCI data traffic produced by zero-copy operations.
- A new scheduler that guarantees fairness in the use of GPU resources. It monitors the pending

[1]University of Málaga, Andalucía Tech, Dept. of Computer Architecture, Spain, e-mail: {alazaro, jgl, blopeza, nguil}@uma.es.

work of each kernel to minimize the kernel slowdown, and implements a policy to balance the global slowdown.

The rest of the article is organized as follows: Section II gives an overview of the preemption mechanism, while the implementation is deeply described in Section III. Section IV introduces our fairness-oriented scheduler along with other popular schedulers used for comparison purpose. Metrics used in our experiments are presented in Section V. The experimental setup and evaluation of our approach are described in Section VI. Related works are discussed in Section VII and, finally, Section VIII concludes the article.

II. Preemption overview

Our scheduler uses the preemption mechanism to implement different QoS-aware scheduling policies. This scheduler, which runs on a CPU thread, is continuously monitoring a queue of eligible kernels. Each kernel is inserted in this queue by the corresponding application. A particular kernel can be declared *eligible* when its required device memory allocation and host-to-device data transfers are done. The application is responsible for memory allocation and the data transfers.

When a condition for eviction is fulfilled, the preemption mechanism is initiated by the scheduler. The scheduler may issue three commands: 1) The *eviction* command indicates that the currently running kernel should be evicted. It is implemented through a data transfer that updates a *State* variable in device memory, which indicates to the running kernel that it has to evict. 2) The *pending work* command is intended to collect information about the kernel remaining work, i.e. the number of pending tasks, by reading a variable located on device memory. 3) The *kernel (re-)launch* command updates the *State* variable associated to the re-launched kernel to set the corresponding value. This way, it guarantees a low delay in the preemption mechanism.

The global memory region used by an evicted kernel is kept during the execution of other kernels. Several works propose different migration techniques to free space in global memory [10]. They are orthogonal to our mechanism, and could be implemented if needed.

Section III gives a detailed description of the implementation of our preemption mechanism while Section IV-E describes our Fair and Responsive Scheduler (FRS), that uses this preemption mechanism to implement a new fairness-aware based scheduling policy.

III. Implementation Details

In this section the two main elements of our preemption mechanism are explained. First, we show the modifications that must be applied to original kernels to support preemption. Second, the core of the preemption mechanism is presented, that is, all the operations involved in the eviction mechanism.

A. Kernel Transformation

The implementation of the preemption mechanism requires the modification of the original kernels, although this modification can be easily automatized. First, the original kernel grid must be modified so that it is executed using persistent thread blocks. Thus, our scheme launches just the number of thread blocks that fit into the available SMs ($maximum\_number\_of\_blocks\_per\_SM \times numSMs$). This transformation can be done automatically by a compiler, as a previous work has shown [7].

The proposed transformation has two advantages. First, persistent thread blocks typically execute more iterations in comparison with no persistent thread blocks and eviction could be performed at the end of each iteration. Thus, when a previously evicted kernel is relaunched, each thread block would just resume the execution from the last executed iteration. Second, this eviction mechanism works faster as only tenths (at most a few hundredths in modern Volta architectures) of persistent thread blocks are active instead of several thousandths blocks in many kernels.

```
1  /************* GPU code *************/
2  Kernel_func (list_of_original_params,
       int MaxNumTask, int *State, int *
       TaskId) {
3    while(true) {
4    // Check state (only one thread)
5      if (threadIdx.x == 0) {
6        if (*State == EVICTED)
7          blockId = -1;
8        else
9          blockId = atomicAdd(TaskId, 1);
10     }
11     // synchronize block threads
12     _sync_threads();
13     // end checking: no more tasks or
           evict
14     if (blockId>=MaxNumTask || blockId
           ==-1)
15       return;
16     // Original kernel code follows
           here
17   }
18 }
19 /********* CPU calling code *********/
20 Kernel_func<persist_grid_size> (
       list_of_original_params, MaxNumTask
       , State, TaskId);
```

Listing 1
Original kernel code modification to support preemption. New code is highlighted in bold type.

A flexible mechanism for load distribution among thread blocks is also proposed. Instead of assigning a specific computation to each thread block with a fixed mapping [11], thread blocks obtain load information by atomically updating a common global memory variable at the start of each iteration. This global memory variable acts as a counter (starting from zero) that increments a task index. These tasks ordering do not affect original kernel execution and benefits the preemption mechanism as only the index of the last executed task must be saved when

a kernel is evicted. In this paper we use the term *task* to name the basic unit of work and it is given by the amount of computation done by one thread block during one iteration.

In Listing 1 main kernel changes are indicated. As it can be observed, two new global memory variables are required per kernel (line 2). One of these variables, called *State*, keeps the state of the GPU kernel. It can take two possible values: Running or Evicted. Just before a kernel is launched it is set to Running. The other memory variable is called *TaskId* and contains the index of the next available task. Initially it is set to 0. In addition there is a new parameter, *MaxNumTask* (also in line 2), that contains the total number of tasks to execute. It is checked by GPU threads to acknowledge when all tasks have been computed and the kernel execution is done.

Listing 1 also shows the code for the eviction mechanism and the tasks distribution strategy. The original thread block computation is enclosed within a while loop (line 4) that finishes when either an eviction command is sent by the CPU scheduler thread or no more pending tasks are available (line 14). At the beginning of each iteration the thread with id 0 is in charge of reading the *State* variable (line 6). If the state has changed to Evicted, a variable mapped in shared memory, called *blockId*, is set to -1 (line 7). This variable is also used to store the task id when the kernel is running (line 9). As it can also be observed in line 9, new values of task id are obtained by thread 0 at each thread block iteration by executing an AtomicAdd instruction on *TaskId* variable.

A block synchronization instruction is also added (line 12) so that the rest of the block threads wait for thread 0. After this barrier, all block threads read *blockId* and check (line 14) finishing condition (all kernel tasks have been computed or *State* has been changed to Evicted). If condition is not fulfilled, thread block executes a new task (with the current *blockId* value). An additional modification must also be applied to the original code to change the indexation employed during computation. In original kernels this indexation is typically implemented using thread block indexes while in the modified kernel the *blockId* variable should be used.

B. Eviction and launching implementation

Unlike previous approaches [7], our preemption mechanism does not require the use of a proxy kernel in the GPU. Therefore, all GPU resources can be devoted to workload execution. In addition, the scheduler can directly access device global memory positions by sending eviction commands and reducing greatly the bus traffic with respect to recent works, [7], [8], as it was explained in Section I.

In previous section we explained that the eviction mechanism requires to distinguish between two kernel states: Running and Evicted. There is a global memory variable, *State*, that contains one of this two possible values. The scheduler is in charge of changing the variable content using a Host to Device (HtD)

Fig. 1. A profiler capture showing the eviction (and re-launching) of five kernels. Three transfers are required to evict a running kernel and launch a new one.

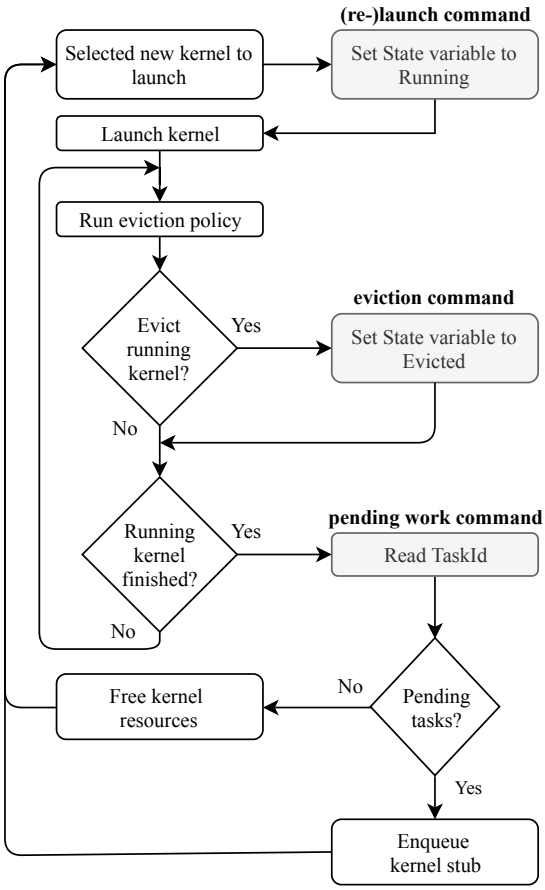


Fig. 2. Flow diagram of the scheduler that illustrates the three transfer commands (grey boxes) involved in the eviction and launching procedures. All commands are sent by the scheduler to TM queues for further processing.

transfer. Thus, as it is shown in Figure 2, before a kernel is launched (or re-launched after a previous eviction) this value is set to Running (*launch command*). When the scheduler decides to evict the running kernel it changes the state to Evict using a new HtD transfer (*eviction command*). Then, when thread blocks start a new iteration, their thread 0 consult the state of this variable and exit if the value is set to Evicted. When all the kernel thread blocks have exited, the kernel is considered evicted. Meanwhile, the scheduler is using a non-blocking synchronization call to detect kernel termination. Kernel termination can happen for two different reasons: 1) all tasks have been executed, or 2) an eviction command has been issued. Thus, the complete eviction process also requires the scheduler checks the number of pending tasks of the just finished kernel. This way the scheduler can know if kernel has finished (no pending tasks) or if it needs to be relaunched later. Thus, when the scheduler detects that the current executing kernel has finished it issues a *pending work*

command to read the current value of the *TaskId* variable.

When a new kernel is selected, the three commands involved in the preemption mechanism are launched again. All these commands are submitted (gray boxes in Figure 2) using specific streams in order to avoid false dependencies with other commands using the same streams.

Figure 1 shows a capture of the NVIDIA profiler, *nvpp*, that illustrates the transfers involved in the preemption mechanism. There are three timelines in the figure. Two of the timelines correspond to HtD and DtH transfers since the GPU used in the experiment has two DMA engines. The other timeline, called Compute, is used to indicate kernel execution. Thus, we can appreciate the execution of several kernels (bars with different colors). In the gap between two kernels execution, a group of three short transfers appears (in brown color). From left to right, the first transfer (HtD) changes the kernel status to Evicted. Then, when kernel finishes, a DtH transfer reads the number of kernel pending tasks. Finally, when a new kernel is selected to run, its status is changed to Running with a HtD transfer.

IV. Schedulers

In this section the schedulers used in our study are presented. All of these schedulers support preemptive scheduling. Thus, when a priority kernel arrives the running kernel is preempted.

A. SJF: Shortest Job First

In Shortest Job First (SJF) the priority of a kernel is based on its duration. The longer its duration, the lower its priority. SJF minimizes the total waiting time of a set of jobs, giving superior responsiveness. For applications running several kernels, their priority will be computed using the total time of their kernels.

B. SRT: Shortest Remaining Time

Shortest Remaining Time (SRT) is similar to SJF but, in this case, the priority is dynamically updated. When a new kernel arrives, the remaining time of each kernel/application is estimated, and a new priority is assigned to each of them. The remaining time of applications running several kernels is calculated using the sum of the remaining times of their kernels.

C. RR: Round Robin

In a Round Robin scheduler (RR) all kernels have the same priority and a fair execution time (*quantum*) is assigned to each kernel. In this way, all kernels will be executed during its *quantum* on a first-come, first-served manner. If a kernel does not finish during its *quantum*, the kernel is preempted and added to the end of the waiting queue.

D. CFS: Completely Fairness Scheduler

The Completely Fair Scheduler (CFS) is a well-known implementation of a fairness policy and it was used as the default scheduler in the Linux kernel (2007 release). CFS defines an *epoch* in order to assign the same amount of computation time at each ready kernel. Thus, the *quantum* assigned to a specific kernel is given by the result of dividing the epoch value by the number of active kernels. Within an epoch, execution ordering is calculated using the time a kernel has been waiting for execution. Thus, kernels are ordered in decreasing order attending to their waiting time values, and scheduled following that ordering.

E. FRS: Fair and Responsive Scheduler

One of the aims of this work is the study of fair policies for scheduling kernels on GPUs. These policies are built using some fairness measure and, in this paper, we have selected the slowdown variance. In this context a fair scheduling policy should try to obtain similar slowdowns for all running jobs, that is, it should implement a *proportional fairness* policy [12]. We consider this type of fairness is interesting for a scheduler that arranges jobs from different applications on a GPU as the assignment of resources among different kernels should tend to be balanced. Because of that, we propose a scheduler, named Fair and Responsive Scheduler (FRS), that uses a kernel slowdown metric to take scheduling decisions.

The design of FRS is based on the instantaneous slowdown, IS, of a kernel. The IS of a GPU kernel that has executed N_t tasks from a total of NT tasks, after spending t seconds since it became ready, is given by

$$IS = (t + (NT - N_t) * tpt)/(N_t * tpt) \qquad (1)$$

being tpt the time, in seconds, required to execute a task. The expression $(NT - N_t)$ is the number of tasks that remains to be executed. Thus, analyzing the expression for IS, we can see that it calculates the rate between the predicted execution time, assuming that all the remaining tasks will be executed in a row, and the shortest execution time, assuming the entire kernel was executed with no interruption. The value of tpt in the expression could be extracted from a brief previous execution of the kernel in the case the kernel has a regular behavior, or it could be updated after each kernel eviction. Notice that the use of the *pending work* command in our preemption mechanism allows the scheduler to know N_t and to calculate tpt at each kernel eviction.

When a kernel is evicted, the scheduler updates the IS values of all the ready kernels and chooses the one with highest IS, IS_{max}. A *quantum* value is assigned to the kernel using the following formula

$$quantum = IS_{max} * NT * tpt - (NT - N_t) * tpt - t \quad (2)$$

where the values of NT, tpt, N_t and t are given by the task with minimum IS value. This way, after executing the chosen kernel for this *quantum*, the new IS value of the task with minimum IS will be the same as the launched task, that is, IS_{max}.

V. Metrics

In this section, we introduce the metrics we have used to measure the effectiveness of each scheduler attending to different criteria as performance, responsiveness and fairness given a set of n tasks.

A. Average normalized turnaround time (ANTT)

The NTT of tasks i is defined as follows:

$$NTT_i = T_i^{MP}/T_i^{SP} \quad (3)$$

where T_i^{MP} and T_i^{SP} are the execution times of the task in its co-run and its standalone run respectively. NTT_i is usually greater than 1, the smaller the more responsive the application is. $ANTT$ is the average of $NTTs$ for all the executed applications.

$$ANTT = \overline{NTT} \quad (4)$$

where NTT is a random variable that takes a specific value for each evaluated task.

B. System overall throughput (STP)

It is defined as follows:

$$STP = \sum_{i=1}^{n} T_i^{SP}/T_i^{MP} \quad (5)$$

STP varies from 0 to n (the number of applications); the higher, the better.

C. Deviation normalized turnaround time (DNTT)

We employ this metric to evaluate how fair is the execution of a set of taks. It is based on the slowdown variance value [12] and considers that a fair scheduling of a set of tasks should obtain low slowdown variance, that is, the lower, the better. Thus,

$$DNTT = \sigma(NTT) \quad (6)$$

VI. Experiments

Experiments have been conducted using different applications with kernels belonging to CUDA SDK [13], Rodinia [14] and Chai [15] benchmark suites. With these applications we pursuit to build a real workload where several applications (up to nine) share the GPU. All experiments have been run on a server with two Xeon(R) E5-2620 CPUs and one NVIDIA TITAN X Pascal. The interconnecting bus is a PCIe 3.0.

Table I shows the list of application used. Most of them have only one kernel, but two of them, namely Separable Convolution and Canny, are composed by two (RCONV and CCONV) and four kernels (GCEDD, SCEDD, NCEDD and HCEDD), respectively. Kernels of both applications are executed in a pipeline fashion since the output of one kernel is the input to the next kernel.

In order to evaluate our approach, several experiments have been designed. Thus, first two experiments focus on analyzing both the overhead of the required kernel modifications and the performance of the preemption mechanism. The third experiment evaluates different fairness scheduling policies.

TABLE I

Applications used in the experiments along with time, in milliseconds, taken by their execution commands. Most of the applications consist of one kernel although Separable Convolution and Canny are composed by two and four kernels, respectively.

Application	Source	Description	Execution Time (ms)
CEDD	Chai	Canny	13.57
SPMV	Rodinia	Sparse MV Mult.	8.36
VA	CUDA SDK	Vector Addition	3.71
CONV	CUDA SDK	Separable Convolution	3.27
BS	CUDA SDK	Black Scholes	2.36
PF	Rodinia	Path Finder	1.65
MM	CUDA SDK	Matrix Mult.	1.43
HST256	CUDA SDK	Histogram	1.29
RED	CUDA SDK	Reduction	0.87

A. Kernel transformation overhead

Implementation of the preemption mechanism requires the transformation of the original kernel code. In this experiment, original and modified kernels are executed until they finish. This way we can measure the overhead incurred by kernel transformation, O_t. The value for this overhead is given by the expression $O_t = \frac{T_t}{T_o}$, where T_t and T_o indicate the execution times of the transformed and original kernels, respectively. Experimental results are shown in the second column of Table II.

Attending to the expression of the overhead, values higher than one are expected since the kernel transformation explained in Section III-A increases the number of instructions executed by the kernel. However, there are cases where values are lower than one. These results can be explained by the fact that kernel transformation also implies a modification in the number and granularity of thread blocks. For instance, original kernel of HST256 uses a small number of blocks with coarse granularity. After our transformation, the number of thread blocks of HST256 are increased and, consequently, the granularity is decreased. With these modifications the kernel runs faster on our device. Focusing on kernels with overhead higher than one we can see that the execution time increases, at most, by 5%, with the exception of MM. Thread blocks of these kernels employ all the available shared memory and the $TaskId$ variable must be mapped in global memory in order to keep the occupancy of the original kernel. Then, the overhead is increased due to longer memory latency. Nevertheless, the average overhead is 0.98, that can be considered almost negligible.

B. Eviction delay

The eviction mechanism is based on asynchronously updating the *State* variable located on GPU global memory (see Sec. III-A) by the scheduler running on the CPU. GPU thread blocks read this variable and, depending on its content, either finish or keep running. Consequently, a delay can be expected between the submission of the transfer command that changes the variable content and the termination of the running kernel. This delay

TABLE II
SECOND COLUMN INDICATES THE OVERHEAD INCURRED BY KERNEL MODIFICATION. THIRD COLUMN SHOWS THE DELAY (IN MICROSECONDS) OF THE PREEMPTION MECHANISM FOR EACH KERNEL.

Kernel	Transformation Overhead	Eviction Delay (μs)
GCEDD	1.02	45
SCEDD	0.98	40
NCEDD	1.02	38
HCEDD	1.04	37
SPMV	0.73	229
VA	1.00	95
BS	0.92	83
RCONV	0.91	53
CCONV	0.97	46
PF	1.00	112
MM	1.11	86
HST256	0.85	94
RED	1.05	32

is directly related to how frequently a thread block reads the variable. Thus, a reduction of the delay requires more frequent memory reads which, however, increase the overhead of the preemption mechanism (as it was discussed in previous subsection). As we are interested in keeping the code modification as simple as possible, the minimum granularity is given by the computation enclosed within a thread block of the original kernel (see Listing 1). If necessary, this granularity could be increased by applying a coarsening technique to the thread block code.

The third column of Table II shows the eviction delay for each kernel. Most of the kernels have an eviction delay lower that 100 μs. Only SPMV has high values for eviction delay. In this kernel, matrix rows are distributed among thread blocks. When the size of the row is large (more precisely, the number of elements different from zero), as in our data set, the computation performed by the thread block is high and the response to changes in the State variable is slow. The average delay for all kernels is around 76μs, a value that permits the development of schedulers with short eviction intervals.

C. Fair scheduling

We have conducted a third set of experiments to compare SJF, SRT, RR, CFS and FRS. Experiments are run by executing all the applications concurrently. Thus, one CPU thread per application is created. All the HtD and DtH commands are launched by these threads while the kernel execution commands are enqueued in a scheduling queue that is managed by the scheduler. Similar to the experiments of other previous works [7], kernels arrive orderly and with 1 ms between them. One hundred of kernels combinations have been run ten times, using each scheduling policy, and their execution times have been averaged by each application and scheduler. In order to obtain more consistent results, all the initial HtD transfers required by the kernels are completed before the scheduler starts.

Several parameters have been also fixed. Thus, the minimum *quantum* size for FRS is established in 1 ms. The epoch size for CFS is fixed to 4 ms (several values were tested and the best value was chosen). The *quantum* size for RR is set to 1 ms and it was chosen to reduce the waiting time without penalizing the turnaround time of short kernels. Table I shows that PF, MM, HST256 and RED have an execution time near to 1 ms; then, if we had used a larger *quantum*, the turnaround time would be affected for these applications.

Although the main objective of this experiment is to compare fairness of different scheduling policies using the DNTT metric, we have also computed the other metrics introduced in Section V to extract complementary information regarding the performance of the evaluated scheduling policies. Thus, Figure 3 shows the normalized turnaround time obtained for each kernel. This metric is used to measure the slowdown of each application. CFS gets the worst NTT, with RR as the second worst scheduler. As we mentioned in Section IV-D, the *quamtum* assigned to each kernel in CFS depends on the active kernels in each epoch. Therefore, the same *quantum* is assigned to short and large kernels, and this increases the turnaround time of the short kernels. RR obtains a better value for *Reduction* because the assigned *quantum* is higher that its execution time thus, although it must wait a long time, it executes completely when it is scheduled. SJF and SRT obtain the best results, followed closely by FRS. Short kernels obtain better values because all these schedulers give them higher priority.

All these NTT values can be used to compute the $DNTT$, $ANTT$ and STP metrics. $DNTT$ metric results are shown in the middle column of Figure 4. A high value means normalized turnaround times of the scheduled applications have a high variability, thus a fair scheduler should obtain low values. CFS and RR get the worst results, with 2.99 and 1.41 respectively, mainly because NTT in short kernels is much larger than in long kernels. SJF and SRT obtain a slowdown variance of less than 1.0, more precisely a value of 0.7 and 0.63 respectively. Finally, FRS obtains the best value (0.42), which is around 1.5, 1.66, 3.35 and 7.11 times lower (better) than SRT, SJF, RR and CFS respectively.

Average values, obtained with the $ANTT$ metric (Figure 4 first column), are related to the responsiveness of the scheduling policy. CFS gets once again the worst value (6.0), followed by RR with a value of 4.87. SJF, SRT and FRS have a similar average normalized response time, close to 2.0 of their standalone application execution time (1.95, 1.96 and 2.44 respectively). These scheduling policies are very responsive, to both short and long kernels.

Finally, we have studied the system overall throughput (STP), showed in Figure 4 third col-

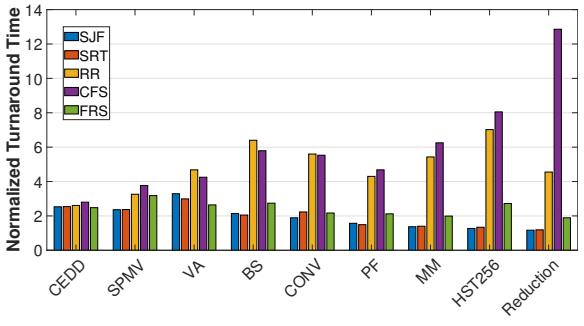


Fig. 3. Normalized turnaround time; SJF, SRT, RR, CFS and FRS priorities are used. The benchmarks are ordered in decreasing order of their simple execution time (from left to right).

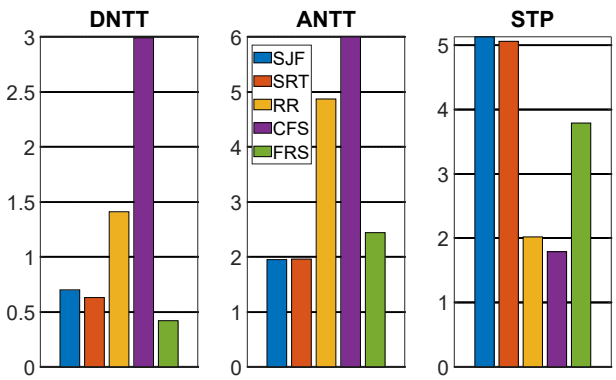


Fig. 4. Average normalized turnaround time ($ANTT$ the lower, the better), deviation normalized turnaround time ($DNTT$ the lower, the better) and system overall throughput (STP the higher, the better) under SJF, SRT, RR, CFS and FRS.

umn. This metric is used to measure the throughput of the system when all the applications are running together with respect to their standalone execution. Thus a higher STP value means a better value. CFS and RR obtain the worst results because the applications need a long time to be executed (1.79 and 2.02 respectively). On the other hand, SFT and SRT get the better STP value with 5.13 and 5.06, respectively, while FRS stands in the middle with a satisfactory result of 3.79.

VII. Related work

Early works on GPU tasks scheduling have tried to increase the responsiveness of the system by minimizing the impact of the non-preemptive nature of both DMA transfers and GPU kernels execution. Thus, Kato et al. [16] studied how to divide memory transfers in chunks to increase the concurrency with kernel launches. Other works, like [3] and [17], have proposed to profile GPU resources to schedule high priority kernels to meet soft real-time requirements. On the other side, some authors have analyzed techniques like elastic kernels [18] or kernel slicing [19], [20], to improve multiple kernels' concurrent execution. Furthermore, some works use the idea of persistent threads [21] for dynamic load balancing of several kernels [22], [23]. Neither of these works have a preemption mechanism that could be used to implement a more sophisticated scheduler or to avoid priority inversion problems.

Recent works have proposed software based preemption mechanisms. Thus, Chen et al. [7] presented a software framework that enables temporal preemption at thread block level. Compared to our work, their mechanism needs a special kernel running in the GPU for each application to serve as a proxy for their host runtime, and lengthy host to/from device transfers can produce priority inversion problems. Wu et al. [8] introduced a spatio-temporal preemption mechanism at thread block level, where thread blocks running in some SMs can be evicted while the others continue to run. Yun et al [24] proposed a similar preemption mechanism, where GPU resources assigned to each kernel are dynamically adjusted by evicting thread blocks. Once again, there is no support for a data transfers control mechanism that could avoid unnecessary delays.

There have been some other works that have studied hardware mechanisms to enable preemptive multiprogramming on GPUs. Tanasic et al. [5] proposed two mechanisms: a context switch that needs to save the execution context of each running thread block, and a SM draining mechanism that waits for each currently running thread block to finish. Park et al. [6] added a third mechanism, SM flushing, to instantly preempt idempotent kernels, i.e., kernels that can be safely restarted. All these works are orthogonal to ours, as an efficient preemption hardware mechanism would benefit our scheduling algorithm.

There are not many works that include QoS or fairness scheduling strategies. Chen et al. [25] presented Prophet, a QoS scheduler that predicts performance of co-located applications. Kerbl et al. [9] proposed bucket queues to schedule tasks using some simple strategies, like FIFO or priorities, or more complex strategies by assigning quotas to the queues. Nevertheless, they do not consider any preemption mechanism thus they are vulnerable to priority inversion problems.

VIII. Conclusions

In this work we have presented a software-based preemption mechanism that can be used to design schedulers for current GPU systems. Experiments with a workload composed by nine applications have shown that the overhead of the kernel modifications required by the preemption mechanism is negligible.

Nevertheless, the main advantage of our scheme, which marks the difference with other recent works, is that a GPU proxy is not needed for the preemption mechanism. Only three transfers are used in our preemption mechanism: 1) an *eviction* command from CPU to GPU, 2) a *pending work* command of the running kernel from GPU to CPU, and 3) a kernel *(re−)launch* command, that updates the *State* variable, from CPU to GPU. Our implementation obtains low delays (less that 0.1 ms) and allows the design of efficient schedulers.

Furthermore, the preemption mechanism also permits to know the pending work of running kernels.

We have used this information to develop a new Fair and Responsive Scheduler, FRS, that tries to balance the instantaneous slowdown of the active kernels. Comparison results with other schedulers, Shortest Job First, Shortest Remaining Time, Round Robin and Completely Fair Scheduler, show that our scheduler obtains the best fairness values, 1.5 times lower (better) than the second best scheduler, using the $DNTT$ metric. Furthermore, FRS get a $ANTT$ of 2.44 close to the most responsive scheduling policies, SJF and SRT. Finally, STP metric shows that FRS gets a satisfactory result of 3.79.

IX. Acknowledgements

This work has been funded by project TIN2016-80920R (Spanish Ministry of Science and Technology) and University of Malaga (Campus de Excelencia Internacional Andalucía Tech). We gratefully acknowledge the support of NVIDIA Corporation with the donation of the TITAN X Pascal GPU used for this research.

Referencias

[1] A.J. J. Lázaro-Muñoz, J.M. González-Linares, J. Gómez-Luna, and N. Guil, "A tasks reordering model to reduce transfers overhead on GPUs," *Journal of Parallel and Distributed Computing*, vol. 109, pp. 258–271, nov 2017.

[2] Florian Wende, Frank Cordes, and Thomas Steinke, "On improving the performance of multi-threaded CUDA applications with concurrent kernel execution by kernel reordering," *Symposium on Application Accelerators in High-Performance Computing*, pp. 74–83, 2012.

[3] Shinpei Kato, Karthik Lakshmanan, Ragunathan Rajkumar, and Yutaka Ishikawa, "Timegraph: Gpu scheduling for real-time multi-tasking environments," in *Proceedings of the 2011 USENIX Conference on USENIX Annual Technical Conference*, Berkeley, CA, USA, 2011, USENIXATC'11, pp. 2–2, USENIX Association.

[4] NVIDIA, "NVIDIA Tesla P100," 2016.

[5] I. Tanasic, I. Gelado, J. Cabezas, A. Ramirez, N. Navarro, and M. Valero, "Enabling preemptive multiprogramming on gpus," in *2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*, June 2014, pp. 193–204.

[6] Jason Jong Kyu Park, Yongjun Park, and Scott Mahlke, "Chimera: Collaborative preemption for multitasking on a shared gpu," in *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems*, New York, NY, USA, 2015, ASPLOS '15, pp. 593–606, ACM.

[7] Guoyang Chen, Yue Zhao, Xipeng Shen, and Huiyang Zhou, "Effisha: A software framework for enabling efficient preemptive scheduling of gpu," in *Proceedings of the 22Nd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, New York, NY, USA, 2017, PPoPP '17, pp. 3–16, ACM.

[8] Bo Wu, Xu Liu, Xiaobo Zhou, and Changjun Jiang, "FLEP: Enabling Flexible and Efficient Preemption on GPUs," in *ASPLOS '17*, New York, New York, USA, 2017, pp. 483–496, ACM Press.

[9] Bernhard Kerbl, Michael Kenzel, Dieter Schmalstieg, Hans Peter Seidel, and Markus Steinberger, "Hierarchical Bucket Queuing for Fine-Grained Priority Scheduling on the GPU," *Computer Graphics Forum*, vol. 36, no. 8, pp. 232–246, 2017.

[10] Taichiro Suzuki, Akira Nukada, and Satoshi Matsuoka, *Euro-Par 2015: Parallel Processing: 21st International Conference on Parallel and Distributed Computing, Vienna, Austria, August 24-28, 2015, Proceedings*, chapter Efficient Execution of Multiple CUDA Applications Using Transparent Suspend, Resume and Migration, pp. 687–699, Springer Berlin Heidelberg, Berlin, Heidelberg, 2015.

[11] Y. Liang, H. P. Huynh, K. Rupnow, R. S. M. Goh, and D. Chen, "Efficient GPU Spatial-Temporal Multitasking," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 3, pp. 748–760, March 2015.

[12] Vincent J. Maccio, Jenell Hogg, and Douglas G. Down, "On slowdown variance as a measure of fairness," *Operations Research Perspectives*, vol. 5, pp. 133 – 144, 2018.

[13] NVIDIA, "Cuda sdk code samples," May 2018.

[14] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S. H. Lee, and K. Skadron, "Rodinia: A benchmark suite for heterogeneous computing," in *Workload Characterization, 2009. IISWC 2009. IEEE International Symposium on*, Oct 2009, pp. 44–54.

[15] J. Gómez-Luna, I. E. Hajj, L. Chang, V. Garca-Floreszx, S. G. de Gonzalo, T. B. Jablin, A. J. Pea, and W. Hwu, "Chai: Collaborative heterogeneous applications for integrated-architectures," in *ISPASS*, April 2017, pp. 43–54.

[16] S. Kato, K. Lakshmanan, A. Kumar, M. Kelkar, Y. Ishikawa, and R. Rajkumar, "RGEM: A Responsive GPGPU Execution Model for Runtime Engines," in *2011 IEEE 32nd Real-Time Systems Symposium*. nov 2011, pp. 57–66, IEEE.

[17] Haeseung Lee and Mohammad Abdullah Al Faruque, "Gpu-evr: Run-time event based real-time scheduling framework on gpgpu platform," in *Proceedings of the Conference on Design, Automation & Test in Europe*, 3001 Leuven, Belgium, Belgium, 2014, DATE '14, pp. 220:1–220:6, European Design and Automation Association.

[18] Sreepathi Pai, Matthew J Thazhuthaveetil, and R Govindarajan, "Improving GPGPU concurrency with elastic kernels," *Proceedings of the eighteenth international conference on Architectural support for programming languages and operating systems - ASPLOS '13*, p. 407, 2013.

[19] Qiumin Xu, Hyeran Jeon, Keunsoo Kim, Won Woo Ro, and Murali Annavaram, "Warped-slicer: Efficient intra-sm slicing through dynamic resource partitioning for gpu multiprogramming," in *Proceedings of the 43rd International Symposium on Computer Architecture*, Piscataway, NJ, USA, 2016, ISCA '16, pp. 230–242, IEEE Press.

[20] J. Zhong and B. He, "Kernelet: High-throughput gpu kernel executions with dynamic slicing and scheduling," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 6, pp. 1522–1532, June 2014.

[21] Timo Aila and Samuli Laine, "Understanding the efficiency of ray traversal on GPUs," *Proceedings of the 1st ACM conference on High Performance Graphics HPG 09*, p. 145, 2009.

[22] Sanjay Chatterjee, Max Grossman, Alina Sbîrlea, and Vivek Sarkar, "Dynamic task parallelism with a GPU work-stealing runtime system," in *7146 LNCS*, 2013, pp. 203–217.

[23] Stanley Tzeng, Brandon Lloyd, and John D. Owens, "A GPU Task-Parallel Model with Dependency Resolution," *Computer*, vol. 45, no. 8, pp. 34–41, aug 2012.

[24] Chao Yu, Yuebin Bai, Hailong Yang, Kun Cheng, Yuhao Gu, Zhongzhi Luan, and Depei Qian, "SMGuard: A Flexible and fine-grained resource management framework for GPUs," *IEEE Transactions on Parallel and Distributed Systems*, 2018.

[25] Quan Chen, Hailong Yang, Minyi Guo, Ram Srivatsa Kannan, Jason Mars, and Lingjia Tang, "Prophet: Precise QoS Prediction on Non-Preemptive Accelerators to Improve Utilization in Warehouse-Scale Computers," *ACM SIGOPS Operating Systems Review*, vol. 51, no. 2, pp. 17–32, apr 2017.

EngineCL: Usability and Performance in Heterogeneous Computing

Raúl Nozal y Jose Luis Bosque[1]

Resumen— **Heterogeneous systems have become one of the most common architectures today, thanks to their excellent performance and energy consumption. However, due to their heterogeneity they are very complex to program and even more to achieve performance portability on different devices. This paper presents EngineCL, a new OpenCL-based runtime system that outstandingly simplifies the execution of a single massive data-parallel kernel on a heterogeneous system. It performs a set of low level tasks regarding the management of devices, their disjoint memory spaces and scheduling the workload between the system devices while providing a layered API. EngineCL has been validated with three devices with different architectures. Experimental results show that it has excellent usability; a negligible overhead compared to the native version, 0.46% on average; and it can reach an average efficiency of 0.89 when balancing the load.**

Palabras clave— **Heterogeneous Computing, Usability, Performance portability, OpenCL, Parallel Programming, Scheduling, Load balancing, Productivity, API**

I. Introduction

The emergence of heterogeneous systems is one of the most important milestones in parallel computing in recent years, mainly due to the computing power and energy efficiency provided by specific purpose hardware accelerators. However, this architecture also presents a series of challenges, among which the complexity of its programming stands out. In this sense, the Open Computing Language (OpenCL) has been developed as an API that extends the C/C++ programming languages for the programming of heterogeneous systems [1]. OpenCL provides low abstraction level that forces the programmer to know the system in detail, determining the architecture of the devices in the heterogeneous system, managing the host-device communication, understanding the distributed address memory space and explicitly partitioning the data among the devices, transferring the input data and collecting the results generated in each device. The management of these aspects greatly complicates programming, which turns into an error-prone process, significantly reducing the productivity [2].

To overcome these problems this paper presents *EngineCL*, a new OpenCL-based C++ runtime API that significantly improves the usability of the heterogeneous systems without any loss of performance. It accomplishes complex operations transparently for the programmer, such as discovery of platforms and devices, data management, load balancing and robustness throughout a set of efficient techniques. EngineCL follows Architectural Principles with known Design Patterns to strengthen the flexibility in the face of changes. Following the Host-Device programming model, the runtime manages a single data-parallel kernel among all the devices in the system.

EngineCL has been validated both in terms of usability and performance, using a system composed of three different architectures: CPU, Xeon Phi and GPU. Regarding usability, 5 metrics have been used, achieving excellent results in all of them. In terms of performance, the runtime overhead compared with OpenCL is on average around 1% when using a single device. Finally, when using all the devices in the system to solve a problem thanks to the provided schedulers, the performance is greatly improved. The geometric mean of efficiencies for the Static algorithm is 0.83 for the regular benchmarks, while for the Dynamic it is up to 0.87 for irregular ones.

The main contributions of this paper are the following:

- Presents EngineCL, a runtime that extraordinarily simplifies the programming of data-parallel application on a heterogeneous system.
- EngineCL ensures performance portability fully exploiting heterogeneous machines.
- An exhaustive experimental validation, both of the usability and the performance of the runtime, proving its excellent behaviour in both metrics.

The rest of this paper is organized as follows. Section II and III describe the design and implementation of EngineCL, respectively. Section IV presents two examples of how to use the API. The methodology used for the validation is explained in Section V, while the experimental results are shown in Section VI. Finally, Section VII explains similar works while Section VIII, the most important conclusions and future work are presented.

II. Principles of Design

EngineCL is designed with many principles in mind, all around three pillars: OpenCL, Usability and Performance.

It is tightly coupled to OpenCL and how it works. The system modules and their relationships have been defined according to the most efficient and stable patterns. Every design decision has been benchmarked and profiled to achieve the most optimal solution in every of its parts, but mainly promoting the modules related with the data management, synchronisation and API abstraction.

[1]Dpto. de Ingeniería Informática y Electrónica, Universidad de Cantabria, e-mail: {nozalr,bosquejl}@unican.es

While OpenCL allows code portability on different types of devices, the programmer is responsible for managing many concepts related to the architecture, such as platforms, contexts, buffers, queues or data transfers. Figure 1 depicts a generic OpenCL program, conceptually and in density of code, compared with the EngineCL version. As the number of devices, operations and data management processes increases, the code grows quickly with OpenCL, decreasing the productivity and increasing the maintainability effort. EngineCL solves these issues by providing a runtime with a higher-level API that efficiently manages all the OpenCL resources of the underlying system.

EngineCL redefines the concept of *program* to facilitate its usage and the understanding of a kernel execution. Because a program is associated with the application domain, it has data inputs and outputs, a kernel and an output pattern. The data is materialised as C++ containers (like *vector*), memory regions (C pointers) and kernel arguments (POD-like types, pointers or custom types). The kernel accepts directly an OpenCL-kernel string, and the output pattern is the relation between the *global work size* and the size of the output buffer written by the kernel. The default value is 1 : 1, because every work-item (thread) writes to a single position in the output buffers ($\frac{1\ out\ index}{1\ work{-}item}$, e.g. the third work-item writes to the third index of every output buffer). It is designed to support massive data-parallel kernels, but thanks to the program abstraction the runtime will be able to orchestrate multi-kernel executions (task-parallelism), prefetching of data inputs, optimal data transfer distribution, iterative kernels and track kernel dependencies and act accordingly. Therefore, the architecture of the runtime is not constrained to a single program.

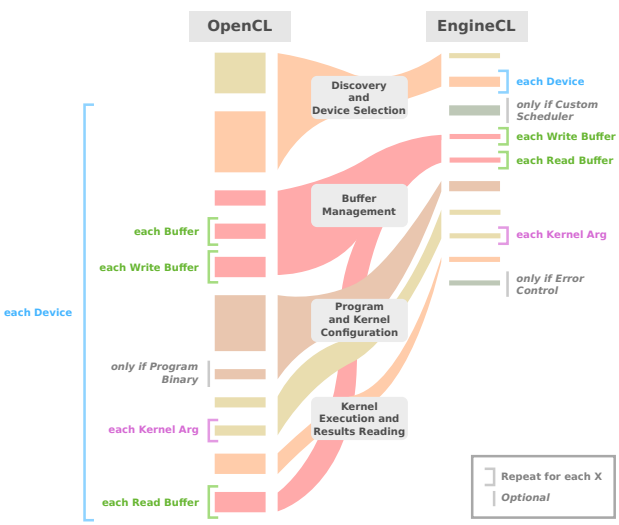


Fig. 1
Overview of a generic OpenCL program and its translation to EngineCL. The height of every rectangle has the same proportions in lines of code as the real program. OpenCL involves more code density and repeats almost all phases per device used.

The runtime follows Architectural Principles with well-known Design Patterns to strengthen the flexibility in the face of changes. As can be seen in Figure 2, the Tier-1 API has been provided mainly because of a Facade Pattern, facilitating the use and readability of the Tier-2 modules, reducing the signature of the higher-level API with the most common usage patterns. The Buffer is implemented as a Proxy Pattern to provide extra management features and a common interface for different type of containers, independently of the nature (C pointers, C++ containers) and its locality (host or device memory). Currently, it supports host-initialised C pointers and C++ vector containers, and other types can be easily integrated with this pattern. Finally, the Strategy Pattern has been used in the pluggable scheduling system, where each scheduler is encapsulated as a strategy that can be easily interchangeable within the family of algorithms. Because of its common interface, new schedulers can be provided to the runtime.

In short, EngineCL is designed following an API and feature-driven development to achieve high external usability (API design) and internal adaptability to support new runtime features when the performance is not penalised. This is achieved through a layered architecture and a set of well-profiled and encapsulated core modules.

III. EngineCL Implementation

EngineCL has been developed in C++, mostly using C++11 modern features to reduce the overhead and code size introduced by providing a higher abstraction level. Many modern features such as *rvalue references*, *initializer lists* or *variadic templates* have been used to provide a better and simpler API while preserving efficient management operations internally. When there is a trade-off between internal maintainability of the runtime and a performance penalty seen by profiling, it has been chosen an implementation with the minimal overhead in performance.

As it is said in Section II and it is depicted in Figure 2, the runtime is layered in three tiers, and its implementation serves the following purposes: Tier-1 and Tier-2 are accessible by the programmer. The lower the Tier, the more functionalities and advanced features can be manipulated. Most programs can be implemented in EngineCL with just the Tier-1, by using the *EngineCL* and *Program* classes. The Tier-2 should be accessed if the programmer wants to select a specific *Device* and provide a specialised kernel, use the *Configurator* to obtain statistics and optimise the internal behaviour of the runtime or set options for the *Scheduler*. Tier-3 contains the hidden inner parts of the runtime that allows a flexible system regarding memory management, pluggable schedulers, work distribution, high concurrency and OpenCL encapsulation.

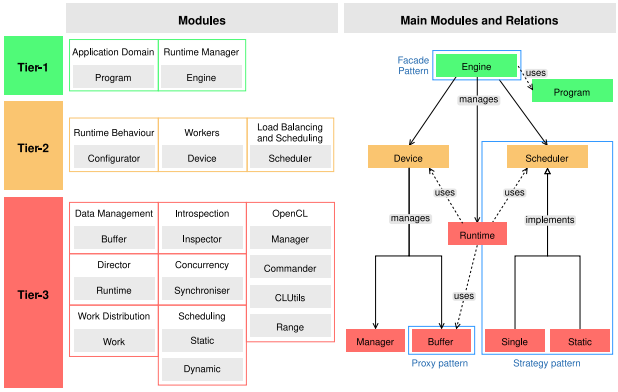


Fig. 2
EngineCL Tiers, main modules and relations.

The implementation follows feature-driven development to allow incremental features based on requested needs when integrating new vendors, devices, type of devices and benchmarks. Implementation techniques are profiled with a variety of OpenCL drivers from the major vendors and versions, but also in devices of different nature, such as integrated and discrete GPUs, CPUs and accelerators. EngineCL has a multi-threaded architecture that combines the best measured techniques regarding OpenCL management of queues, devices and buffers. Some of the decisions involve atomic queues, parallelised operations, custom buffer implementations, reusability of costly OpenCL functions, efficient asynchronous enqueueing of operations based on callbacks and event chaining. These mechanisms are used internally by the runtime and hidden from the programmer to achieve efficient executions and transparent management of devices and data.

The EngineCL architecture allows to easily incorporate a set schedulers, as it is shown in Figure 2. In this paper, two well-known load balancing algorithms are implemented in EngineCL [2]. The programmer should decide which one to use in each case, depending on the characteristics and knowledge he has of the architecture. The algorithms used in this article are briefly described:

.0.a Static. This algorithm works before the kernel is executed by dividing the data-set in as many packages as devices are in the system. The division relies on knowing the percentage of workload assigned to each device, in advance. Then the execution time of each device can be equalized by proportionally dividing the data-set among the devices. It minimizes the number of synchronization points; therefore, it performs well when facing regular loads with known computing powers that are stable throughout the data-set. However, it is not adaptable, so its performance might not be as good with irregular loads.

.0.b Dynamic. It divides the data-set in a given number of equal-sized packages. The number of packages is well above the number of devices in the heterogeneous system. During the execution of the kernel, a master thread in the host assigning packages to the different devices, including the CPU. This algorithm adapts to the irregular behaviour of some applications. However, each completed package represents a synchronization point between the device and the host, where data is exchanged and a new package is launched. This overhead has a noticeable impact on performance if the number of packages is high.

IV. API Utilisation

Listings 1 and 2 depict two benchmark examples using the EngineCL runtime, Binomial and NBody. Both programs start reading their kernels, defining variables, containers (C++ *vectors*) and OpenCL values like *local* and *global work size* (`lws`, `gws`). Then, each program is initialised based on the benchmark (`init_setup`), in lines 9 and 13 (*L9,L13*), respectively. The rest of the program is where EngineCL is instantiated, used and released.

Regarding the Binomial example, the `engine` *uses* the first CPU in the system by using a `DeviceMask` (*12*). Then, the *gws* and *lws* are given by explicit methods (*L14,15*). The application domain starts by creating the `program` and setting the input and output containers with methods `in` and `out` (*L17-19*). With these statements the runtime manages and synchronises the input and output data before and after the computation. The `out_pattern` is set because the implementation of the Binomial OpenCL kernel uses a writing pattern of $\frac{1\ out\ index}{255\ work-items}$ (*L21*), that is, 255 work-items compute a single out index. Then, the kernel is configured by setting its source code string, name and arguments. Assignments are highly flexible, supporting aggregate and positional forms, and above all, it is possible to transparently use the variables and native containers (*L23-29*). The enumerated `LocalAlloc` is used to determine that the value represents the bytes of local memory that will be reserved, reducing the complexity of the API. Finally, the runtime consumes the program and all the computation is performed (*L32,34*). When the `run` method finishes, the output values are in the containers. Optionally, errors can be checked and processed easily.

NBody program shows a more advanced example where EngineCL really excels. Three kernels are used: a common version, a specific implementation for GPUs and a binary kernel built for the Xeon Phi (*L1-3*). The `Device` class from the Tier-2 allows more features like platform and device selection by index (`platform, device`) and specialisation of kernels and building options. Three specific devices are instantiated, two of them with special kernels (source and binary) by just giving to them the file contents (*L17*). After setting the work-items in a single method, the runtime is configured to use the `Static` scheduler with different work distributions for the CPU, Phi and GPU (*L23,24*). Finally,

```
auto kernel = file_read("binomial.cl");
auto samples = 16777216; auto steps = 254;
auto steps1 = steps + 1; auto lws = steps1;
auto samplesBy4 = samples / 4;
auto gws = lws * samplesBy4;
vector<cl_float4> in(samplesBy4);
vector<cl_float4> out(samplesBy4);

binomial_init_setup(samplesBy4, in, out);

ecl::EngineCL engine;
engine.use(ecl::DeviceMask::CPU); // 1 Chip

engine.global_work_items(gws);
engine.local_work_items(lws);

ecl::Program program;
program.in(in);
program.out(out);

program.out_pattern(1, lws);

program.kernel(kernel, "binomial_opts");
program.arg(0, steps); // positional by index
program.arg(in); // aggregate
program.arg(out);
program.arg(steps1 * sizeof(cl_float4),
            ecl::Arg::LocalAlloc);
program.arg(4, steps * sizeof(cl_float4),
            ecl::Arg::LocalAlloc);

engine.use(std::move(program));

engine.run();

// if (engine.has_errors()) // [Optional lines]
//   for (auto& err : engine.get_errors())
//     show or process errors
```

Listing 1: EngineCL API used in Binomial benchmark.

```
auto kernel = file_read("nbody.cl");
auto gpu_kernel = file_read("nbody.gpu.cl");
auto phi_kernel_bin =
  file_read_binary("nbody.phi.cl.bin");
auto bodies = 512000; auto del_t = 0.005f;
auto esp_sqr = 500.0f; auto lws = 64;
auto gws = bodies;
vector<cl_float4> in_pos(bodies);
vector<cl_float4> in_vel(bodies);
vector<cl_float4> out_pos(bodies);
vector<cl_float4> out_vel(bodies);

nbody_init_setup(bodies, del_t, esp_sqr, in_pos,
                 in_vel, out_pos, out_vel);

ecl::EngineCL engine;
engine.use(ecl::Device(0, 0),
           ecl::Device(0, 1, phi_kernel_bin),
           ecl::Device(1, 0, gpu_kernel));

engine.work_items(gws, lws);

auto props = { 0.08, 0.3 };
engine.scheduler(ecl::Scheduler::Static(props));

ecl::Program program;
program.in(in_pos);
program.in(in_vel);
program.out(out_pos);
program.out(out_vel);

program.kernel(kernel, "nbody");
program.args(in_pos, in_vel, bodies, del_t,
             esp_sqr, out_pos, out_vel);

engine.program(std::move(program));

engine.run();
```

Listing 2: EngineCL API used in NBody benchmark.

the program is instantiated without any out pattern, because every work-item computes a single output value, and the seven arguments are set in a single method, increasing the productivity even more (*L33*).

As it is shown, EngineCL manages both programs with an easy and similar API, but completely changes the way it behaves: Binomial is executed completely in the CPU, while NBody is computed using the CPU, Xeon Phi and GPU with different kernel specialisations and workloads. Platform and device discovery, data management, compilation and specialisation, synchronisation and computation are performed transparently in a few lines for the programmer.

V. Methodology

EngineCL has been validated both in terms of usability and performance. Five benchmarks have been used to show a variety of scenarios regarding the ease of use, overheads compared with a native version in OpenCL C++ and performance gains when multiple heterogeneous devices are co-executed. Table I shows the properties of every benchmark. Gaussian, Binomial, Mandelbrot and NBody are part of the AMD APP SKD, while Ray is an open source implementation of a Raytracer. Three different ray-tracing scenes with different complexities are provided to be benchmarked when doing the load balancing. These five benchmarks are selected because they provide enough variety in terms of OpenCL development issues, regarding many parameter types, local and global memory usage, custom structs and types, number of buffers and arguments, different local work sizes and output patterns.

The validation of usability is performed with five metrics based on a set of studies ([3], [4], [5], [6]). These metrics determine the usability of a system and the programmer productivity, because the more complex the API is, the harder it is to use and maintain the program.

The *McCabe's cyclomatic complexity* (*CC*) measures the number of linearly independent paths. It is the only metric that is better the closer it gets to 1, whereas for the rest a greater value supposes a greater complexity. The *number of C++ tokens* (*TOK*) and *lines of code* (*LOC*, via *cloc*) determine the amount of code. The *Operation Argument Complexity* (*OAC*) gives a summation of the complexity of all the parameters types of a method, while *Interface Size* (*IS*) measures the complexity of a method based on a combination of the types and number of parameters. A ratio of $\frac{OpenCL}{EngineCL}$ is calculated to show the impact in usability per benchmark and metric.

Regarding the performance evaluation, the experiments are carried in a machine composed of two Intel

TABLA I
BENCHMARKS AND VARIETY OF PROPERTIES USED IN THE VALIDATION.

Property	Gaussian	Ray	Binomial	Mandelbrot	NBody
Local Work Size	128	128	255	256	64
Read:Write buffers	2:1	1:1	1:1	0:1	2:2
Out pattern	1:1	1:1	1:255	4:1	1:1
# kernel args	6	11	5	8	7
Use local memory	no	yes	yes	no	no
Use custom types	no	yes	no	no	no

TABLA II
COMPARISON OF USABILITY METRICS FOR A SET OF PROGRAMS IMPLEMENTED IN OPENCL AND ENGINECL.

Program	Runtime	CC	TOK	OAC	IS	LOC
Gaussian	OpenCL	4	585	312	433	87
	EngineCL	1	60	33	53	15
	ratio	4	9.8	9.5	8.2	5.8
Ray	OpenCL	4	618	307	424	89
	EngineCL	1	191	40	65	24
	ratio	4	3.2	7.7	6.5	3.7
Binomial	OpenCL	4	522	255	355	77
	EngineCL	1	81	28	48	18
	ratio	4	6.4	9.1	7.4	4.3
Mandelbrot	OpenCL	4	473	222	313	74
	EngineCL	1	65	35	55	15
	ratio	4	7.3	6.3	5.7	4.9
NBody	OpenCL	4	658	373	517	96
	EngineCL	1	66	38	60	16
	ratio	4	10.0	9.8	8.6	6.0
	ratio	**4**	**7.3**	**8.5**	**7.3**	**4.9**

Xeon E5-2620 CPUs, a NVIDIA Kepler K20m GPU and an Intel Xeon Phi KNC 7120P. Thanks to the QPI connection the CPUs are treated as a single device, and it is so by the OpenCL Driver.

Regarding the EngineCL performance two types of experiments are presented. The first measures the overhead of EngineCL compared with OpenCL C++ when using a single device. Every benchmark has five custom problem sizes per device, each one with completion times around 5, 10, 15, 20 and 25 seconds. These comparisons are performed against ranges of completion times due to the heterogeneity of the three devices: limits regarding memory, computing power and global work size.

The second analyses the co-execution performance when using different scheduling configurations in a heterogeneous system composed of three different devices. Each program uses a single problem size, given by the completion time of 15 seconds in the fastest device (GPU).

To guarantee integrity of the results in both experiments, 60 executions are performed per case, divided in three sets of no consecutive executions. Every set of executions performs 20 iterations contiguously without a wait period. An initial execution is discarded for every set of iterations to avoid warm-up penalties in some OpenCL drivers and devices.

Three metrics are used to evaluate the performance of EngineCL: time overhead for the first experiment as well as speedup and efficiency for the second one.

The time overhead, expressed as percentage, is computed as the ratio between the difference of the response times in the execution of the same kernel for both EngineCL (T_{ECL}) and the native version (T_{OCL}), as follows: $Overhead = \frac{T_{ECL} - T_{OCL}}{T_{OCL}} \cdot 100$.

To evaluate the performance of the load balancing algorithms the total response time is measured, including the device initialisation and management, input data and results communications. Therefore, the worst case scenario is considered against a single device execution. From that, the speedup is calculated as the ratio between the execution time on the GPU and on the heterogeneous system. Due to the heterogeneity of the system and the different behaviour of the benchmarks, the maximum achievable speedups depend on each benchmark. These values were derived from the response time T_i of each device:

$$S_{max} = \frac{1}{max_{i=1}^{n}\{T_i\}} \sum_{i=1}^{n} T_i \qquad (1)$$

Additionally, the efficiency of the heterogeneous system has been computed as the ratio between the maximum achievable speedup and the empirically obtained speedup for each benchmark. $Eff = \frac{S_{real}}{S_{max}}$.

VI. VALIDATION

A. Usability

This section shows the experiments performed to evaluate the usability introduced by EngineCL when a single device is used. Table II presents the values obtained for every benchmark (rows) in every of the five metrics (columns). Also, the average ratio per metric is presented.

For every program, the maintainability and testing effort is reduced drastically, as can be seen with *CC*, reaching the ideal cyclomatic complexity.

The density of the code and complexity of the operations involved is reduced between 7.3 to 8.5 times compared with OpenCL, as it is shown with TOK, OAC and IS. In programs like Ray the ratio for *OAC* is greater than in *TOK*, because the amount of parameters grows in both implementations, but managing complex types is harder in OpenCL.

The number of classes instantiated and used methods are around 5 and 2 times less than in the OpenCL implementation, mainly because it has been deliberately instantiated the *Device* and one argument per line is used (`program.arg`), instead of using `DeviceMask` to avoid direct instantiations and a more contract specification of arguments in a single line (`program.args`).

As a summary, EngineCL has excellent results in maintainability, implying less development effort. Thanks to its API usability, the programmer is able to focus on the application domain, and its produc-

tivity is boosted by hiding complex decisions, operations and checks related with OpenCL.

B. Performance

This section presents results of experiments performed to evaluate the overhead introduced by EngineCL when a single kernel is executed in a single device and when using all the devices in the system with the Static and Dynamic schedulers. Figure 3 shows the overhead measured, comparing the OpenCL and EngineCL completion time for each problem size, in percentage. Each row presents the results of a different device, CPU, Xeon Phi and GPU, while each column corresponds to a benchmark. Five results are shown per benchmark, each one with a different problem size as it was explained in Section V. It should be noted that negative overhead values indicate that running with EngineCL is more efficient (uses less time) than running natively with OpenCL.

Analysing each device separately, it can be observed that the worst results are obtained in the CPU, with an average (geometric mean) overhead of 0.46% and a maximum of 2.83% in NBody. This is reasonable since EngineCL also runs on the CPU, so it interferes with the execution of benchmarks, stealing them computing capacity. Regarding the discrete devices, the Xeon Phi presents the best results with an average overhead 0.18%. Finally, the results achieved with the GPU are also excellent, with an average overhead of 0.26% and a maximum value of 1.46%. The differences with GPU and Xeon Phi are explained by the different implementation of the OpenCL driver and how it is affected by the multithreaded and optimised architecture of EngineCL.

The performance results achieved in the heterogeneous system with different load balancing algorithms are shown in Figures 4 and 5, where the speedups and efficiency are depicted, respectively.

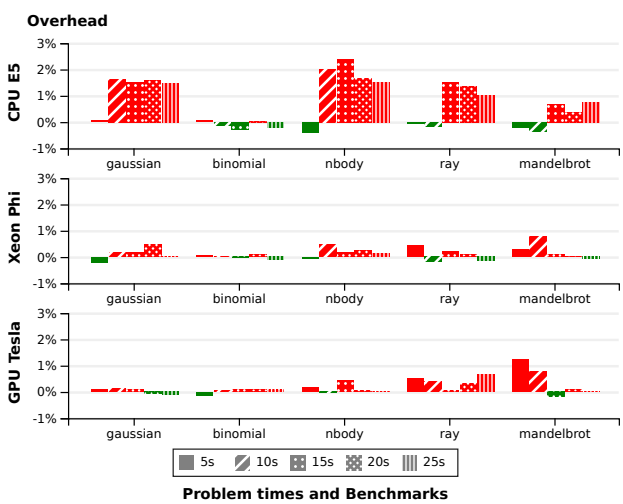


Fig. 3
Overheads in CPU Intel Xeon E5, Xeon Phi and NVIDIA GPU Tesla.

The abscissa axis shows the speedup and efficiency of the heterogeneous system for every scheduling configuration and benchmark. The ordinate axis contains the benchmarks defined in Section V, each one with five scheduling configurations. The first two bars represent the Static algorithm varying the order of delivering the packages to the devices. The one labelled *Static* delivers the first chunk to the CPU, the second to the Phi and the last one to the GPU, while in the *Static rev* the order is *gpu-phi-cpu*. The latter three show the Dynamic scheduler configured to run with 50, 100 and 150 chunks. These variations are presented to show the behaviours of the algorithms in different scenarios.

The speedups presented in this section are due to the co-execution of the benchmarks on the CPU, Xeon Phi and GPU simultaneously, compared with only using the fastest device, the GPU. The efficiency gives an idea of how well a load is balanced. A value of 1.0 represents that all the devices have been working all the time, thus achieving the maximum obtainable speedup. The figures reveal that, for all benchmarks, there is at least one algorithm that offers excellent results with an efficiency of at least 0.84, except for Mandelbrot, where the efficiency is up to 0.76. Therefore, EngineCL can adapt to different kinds of loads obtaining outstanding performance.

Analysing the speedups and efficiencies in detail, it can be seen that, Static delivers good results in regular applications, with consistent efficiencies between 0.80 and 0.87, regardless of the order of the devices. Binomial is an exception that will be explained later. However, in irregular applications the results are much more erratic, because it does not adapt to these irregularities. For instance, there is a strong difference in efficiency between Ray1 (0.78) and Ray2 (0.92), being the same benchmark, but with different distribution of the input data. Furthermore, the order in which the devices are considered also has a significant impact on efficiency, as it is shown in Ray2 and Mandlebrot. The Mandelbrot set is irregular because it grows horizontally and the fractals are displaced to the bottom, having white areas in the top of the image. Therefore, when the slowest device (CPU) processes the first part of the image (chunk 1 in *static*) it has better speedup than when it processes the last part (chunk 3 in *static rev*), unbalancing the execution.

The Dynamic algorithm excels in most applications, achieving a geometric average efficiency of 0.81, but suffers in benchmarks like NBody and Gaussian. The former shows the overhead of communication when using many chunks and the latter the execution imbalance when using a few chunks, increasing the work size and penalising the balancing due to slow devices. Therefore, it is important to accurately determine the number of packages to get the best results in each benchmark.

Figure 6 depicts the work size distribution between the devices for every scheduler configuration and benchmark. Each bar has three rectangles with

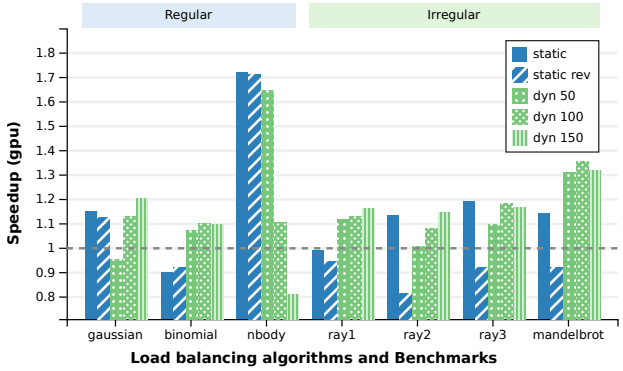


Fig. 4
Speedups for every scheduler compared with a single GPU.

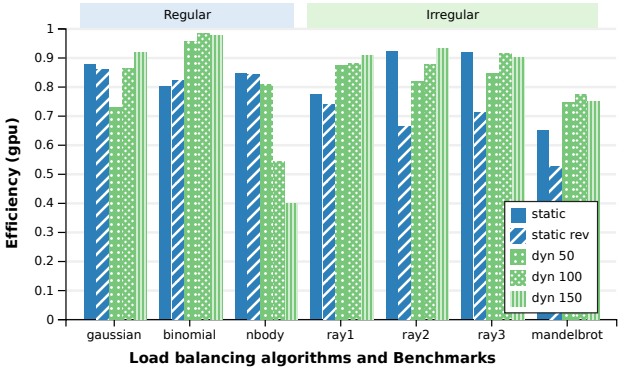


Fig. 5
Efficiency of the system when adding the CPU and Xeon Phi.

the work size given to each device. In the Static approach, the size is given by the proportions of computing power calculated offline, while in the Dynamic it is the final work size given at runtime.

Every scheduling configuration distributes a similar workload for each device, except in NBody and Mandelbrot. The CPU takes more workload as the number of packages increases in NBody, introducing synchronisation overheads that are negligible when fewer packages are used. On the other side, Mandelbrot shows how the Phi processed too much amount of work for the part of the image given in the Static, being more complex to calculate than the expected when computing the complete image.

As it was introduced, the GPU in Binomial outperforms the CPU and Xeon Phi, as can be seen in the Static work size distributions. Therefore, a slightly variation in the completion time for any of these devices will imbalance the execution. Another important point is introduced to the analysis: the implementation of the OpenCL driver of the Xeon Phi needs and uses the CPU for its management. When using the CPU in co-execution, the Xeon Phi driver needs to share the CPU to build and manage the device with the CPU OpenCL driver, introducing time variations during the initialisation and new overheads in the final completion times. This behaviour is depicted in Figure 7, showing the average times from initialisation for all the executions in Binomial, where the abscissa axis shows the base case (single device) and each scheduling configuration, with a bar showing the behaviour for each device. The ordinate axis shows the time since EngineCL started. Using only the Phi needs 1000 ms. to initialise and start computing, while it is up to 3000-4000 ms. when using in Static. This variation combined with the small amount of work given to the CPU and Phi produces enough imbalance to not achieve the goal. On the other side, the Dynamic approach it is much worth for two reasons: it allows small periods of CPU time without computation (between chunks) to the Phi driver and thanks to its adaptability solves the initialisation variations giving more chunks to the GPU, as it is shown.

In summary, we can conclude that EngineCL can execute a single massive data-parallel kernel simultaneously on all devices in a heterogeneous system with negligible overhead. In addition, thanks to the load balancing algorithms, it yields excellent efficiencies, with different types of benchmarks.

VII. Related Work

There are projects aiming at high-level parallel programming in C++, but most of them provide an API similar to the Standard Template Library (STL) to ease the parallel programming, like

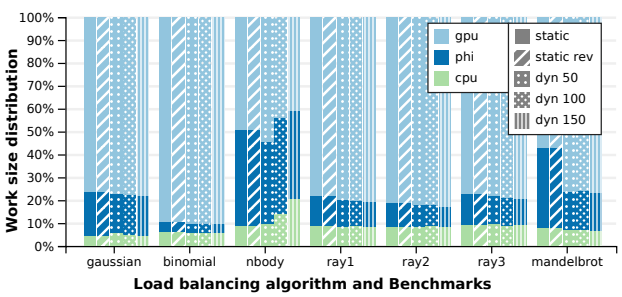


Fig. 6
Work size distribution per device and benchmark.

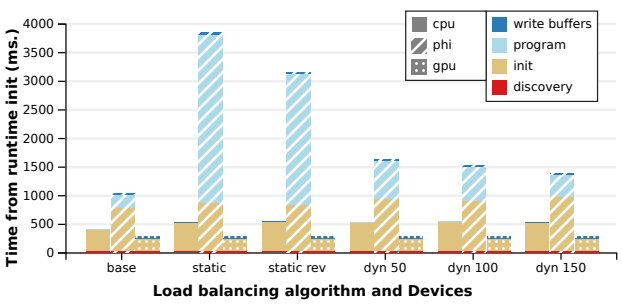


Fig. 7
Binomial timings previous to the computation phase.

Boost.Compute [7], HPX [8], Thrust [9], SYCL [10] and the C++ Extensions for Parallelism TS [11]. Thrust, tied to CUDA devices, HPX, which extends the C++ Parallel and Concurrency TS with future types for distributed computing, or C++ TS are not OpenCL-centered. Projects like HPX.Compute [12] and SYCLParallelSTL [13] provide backends for OpenCL via SYCL. SYCLParallelSTL exposes ParallelSTL on CPU and GPU. Proposals like SkelCL [14] and SkePU [15] provide data management and composable primitives and skeletons to build parallel applications, but the programmer is responsible of using their own data containers. EngineCL offers a high-level layered API with better usability than the previous C++ proposals, generally provide constructs based on STL. Also, there are C-programmed libraries with similar objectives, but they provide low-level APIs where the programmer needs to specify many parameters and the density of the code is considerable. While Maat [2] uses OpenCL to achieve the code portability, Multi-Controllers [16] is CUDA and OpenMP-centered, but allows kernel specialisation. On the other side, EngineCL targets a flexible API with an application domain as execution unit, increasing significantly the productivity. It provides different API layers, allows kernel specialisation, direct usage of C++ containers, manages the data and work distribution transparently between devices and has negligible overheads compared with the previous projects.

VIII. Conclusions and Future Work

Given the great relevance of heterogeneous systems in all sectors of computing, it is necessary to provide the community with tools that facilitate their programming, while maintaining the same performance. For this purpose, EngineCL is presented, a powerful OpenCL-based tool that greatly simplifies the programming of applications for heterogeneous systems. This runtime frees the programmer from tasks that require a specific knowledge of the underlying architecture, and that are very error prone, with a great impact on their productivity.

The API provided to the programmer is very simple, thus improving the usability of heterogeneous systems. This statement is corroborated by the exhaustive validation that is presented, with a large variety of Software Engineering metrics, achieving excellent results in all of them. On the other hand, the careful design and implementation of EngineCL allows zero overheads with respect to the native OpenCL version in some experiments. In the rest of the cases, the overhead due to the management performed by EngineCL is negligible, always below 3% in all the cases studied and with an average overhead of 0.46%, achieving an excellent portability performance. Also, two load balancing algorithms have been implemented and validated in order to give the best performance to both regular and irregular applications. The use of the whole heterogeneous system is always beneficial for at least one load balancing method, achieving an average efficiency of 0.89 when selecting the best scheduling configuration. This excellent performance, together with its proven usability, makes EngineCL a very powerful tool for exploiting all kind of heterogeneous systems.

In the future, it is intended to extend the API to support iterative and multi-kernel executions. Also, new load balancing algorithms will be provided and studied as part of the scheduling system, focusing on performance and energy efficiency.

Acknowledgement

This work has been supported by the the Spanish Ministry of Education (FPU16/ 03299 grant), the Spanish Science and Technology Commission (TIN2016-76635-C2-2-R), the European Union's Horizon 2020 research and innovation programme and HiPEAC Network of Excellence (Mont-Blanc project under grant 671697).

Referencias

[1] Gaster, B.R. et al. : Heterogeneous Computing with OpenCL - Revised OpenCL 1.2 Edition, Morgan Kaufmann (2013)
[2] Pérez, B. et al. : Simplifying programming and load balancing of data parallel applications on heterogeneous systems. Proc. of the 9th Workshop on General Purpose Processing using GPU (2016)
[3] De Souza, C.R. et al: Automatic evaluation of API usability using complexity metrics and visualizations. 31st Int. Conf. Software Engineering, ICSE (2009)
[4] Bandi, R.K. et al.: Predicting maintenance performance using object-oriented design complexity metrics. IEEE Transactions on Software Engineering (2003)
[5] Rama, G.M. et al.: Some structural measures of API usability. Software - Practice and Experience (2013)
[6] Scheller, T. et al.: Automated measurement of API usability: The API Concepts Framework. Information and Software Technology (2015)
[7] Szuppe, J.: Boost.compute: A parallel computing library for C++ based on OpenCL. Int. Workshop on OpenCL (2016), accessed on Feb 2018
[8] Heller, T. et al. : HPX – An open source C++ Standard Library for Parallelism and Concurrency. Workshop on Open Source Supercomputing (2017)
[9] Hoberock, J. et al.: Thrust: A Parallel Template Library for C++ (2009)
[10] Group, T.K. et al.: SYCL: C++ Single-source Heterogeneous Programming for OpenCL. SYCL 1.2.1 Specification, accessed on Feb 2018
[11] ISO/IEC: Technical Specification for C++ Extensions for Parallelism (2015)
[12] Copik, M. et al.: Using SYCL As an Implementation Framework for HPX.Compute. Proceedings of the 5th Int. Workshop on OpenCL, IWOCL (2017)
[13] Ruyman Reyes et al.: SyclParallelSTL: Implementing ParallelSTL using SYCL. Int. Workshop on OpenCL (2015), accessed on Feb 2018
[14] Steuwer, M. et al.: SkelCL - A portable skeleton library for high-level GPU programming. IEEE IPDPSW (2011)
[15] Enmyren, J. et al.: SkePU: A multi-backend skeleton programming library for multi-gpu systems. Proc. 4th Int. Workshop on High-Level Parallel Programming and Applications (2010)
[16] Moreton-fernandez, A. et al. : Multi-Device Controllers : A Library To Simplify The Parallel Heterogeneous Programming. Int. J. of Parallel Programming (2017)

Transferencias de datos asíncronas y transparentes en plataformas heterogéneas

Victor Lara-Mongil[1], Ismael Taboada-Rodero[1], Eduardo Rodriguez-Gutiez[2], Yuri Torres[2], Arturo Gonzalez-Escribano[2] y Diego R. Llanos[2]

Resumen— **Los coprocesadores de alto rendimiento, como las Unidades de Procesamiento Gráfico (GPUs), presentan un ratio alto entre rendimiento y coste junto con un bajo consumo de energía. Por ello, los sistemas heterogéneos que los incluyen han experimentado un crecimiento significativo. Sin embargo, la programación de estos dispositivos sigue suponiendo un reto. Uno de los problemas está relacionado con la gestión de la memoria. Estos dispositivos tiene su propio espacio de memoria y es necesario realizar costosas transferencias de datos entre la máquina anfitriona y el dispositivo.**

En este trabajo proponemos una novedosa solución en tiempo de ejecución que analiza las dependencias de las diferentes transferencia de datos, ejecución de kernels y operaciones de host, solapándolas, en la medida de lo posible, de forma automática. Esta solución puede ocultar las latencias de forma transparente, mejorando significativamente el rendimiento de la aplicación. La técnica propuesta está implementada en el modelo de programación de Controllers para plataformas heterogéneas.

Presentamos un estudio experimental que compara programas desarrollados utilizando nuestra solución con programas desarrollados con CUDA y OpenCL. Las versiones implementadas consideran tanto transferencias síncronas como asíncronas. El estudio muestra que la abstracción propuesta introduce un sobrecoste despreciable, mientras que mejora el tiempo de ejecución y reduce el esfuerzo de desarrollo del programa, evitando el uso explícito de mecanismo de asincronía. Los resultados ofrecen hasta un 44.6 % de reducción del tiempo de ejecución de una aplicación real de retransmisión de vídeo, debido al solapamiento de las transferencias de datos y la ejecución de los kernels.

Palabras clave— **Computación heterogénea, sistemas en tiempo de ejecución, ocultación de latencia, ejecución asíncrona.**

I. Introducción

LOS coprocesadores de alto rendimiento, como las Unidades de Procesamiento Gráfico (GPUs), han sido ampliamente adoptadas en los sistemas modernos para la computación de propósito general. Esto se refleja en la configuración de muchos supercomputadores en las posiciones más altas del ranking TOP500 [1]. Sin embargo, el esfuerzo necesario para la explotación eficiente de estos modelos de programación para estos dispositivos, como OpenCL, CUDA, o OpenMP, sigue siendo un desafío.

Muchos modelos de programación para plataformas heterogéneas y coprocesadores usan el concepto de *kernel*. Esta es una unidad de código que se compila para un dispositivo de procesamiento específico, cuya ejecución puede ser solicitada por el programa principal. Los procesadores multi- o many-core explotan los recursos hardware subyacentes para paralelizar la ejecución de los kernel, pero estos deben ser lanzados, ejecutados y sincronizados como uno solo. Las transferencias de datos entre las jerarquías de memorias de la máquina anfitriona, en adelante máquina *host*, y el coprocesador, se expresan en un grado grueso. La petición de estas transferencias normalmente implican el movimiento de estructuras de datos completas que minimicen los costes asociados a las operaciones de transferencia individuales. Los programas están estructurados como una secuencia de llamadas a interfaces de programación específicas, mezclando transferencias de datos y lanzamiento de kernels con código ejecutado en el host. Cuando estas operaciones se ejecutan en orden, la semántica se mantiene sin alteraciones. En cambio, las operaciones de transferencia de datos y los kernels son costosas, y su ejeución síncrona produce retrasos importantes en el camino crítico del programa (ver ejemplo de la figura 1).

Para solventar este problema, muchos coprocesadores e interfaces de programación permiten el solapamiento de computación y transferencias de memorias mediante el uso de operaciones asíncronas. La solución a este conocido problema de solapamiento de transferencias y computación [2] supone un desarrollo tedioso y propenso a errores. Ello implica el uso de conceptos complejos y funcionalidades específicas soportadas por la interfaz de programación (como *streams* o *events*), la creación de funciones *callback*, y el uso de mecanismos de sincronización. A su vez, este desarrollo requiere el análisis del programa para detectar aquellas situaciones donde se pueden explotar las operaciones asíncronas sin afectar al comportamiento del programa.

En este trabajo proponemos una solución para aprovechar e introducir ejecuciones asíncronas en secuencias genéricas de transferencia de datos, ejecuciones de código host y computación de kernels, en diferentes tipos de coprocesadores. Nuestra técnica analiza en tiempo de ejecución la secuencia de operaciones solapando, de forma automática, la transferencia de datos y la computación siempre que sea posible. La solución está implementada como una política de gestión de cola de trabajos en el modelo de programación de Controllers para plataformas heterogéneas [3], [4]. La interfaz y funcionalidades del Controller han sido rediseñadas para proveer una librería genérica para el uso de operaciones básicas que han sido consideradas en esta propuesta. Esta nueva

[1]Dpto. de Informática, Univ. de Valladolid, e-mail: {victor.lara|ismael.jose.taboada}@uva.es.

[2]Dpto. de Informática, Univ. de Valladolid, e-mail: {eduardo|yuri.torres|arturo|diego}@infor.uva.es.

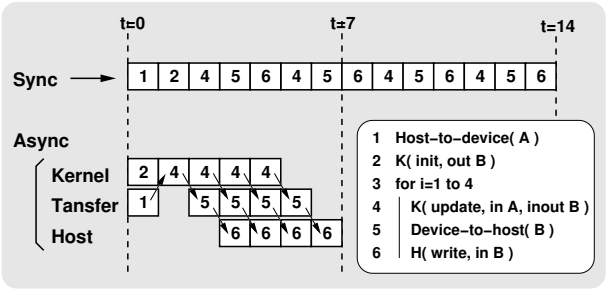


Fig. 1. Ejemplo de modos de ejecución síncrona y asíncrona de una secuencia de operaciones. El modelo asíncrono permite el solapamiento de las transferencias de datos con la ejecución de kernels en el dispositivo K, y con la ejecución de funciones en el host H. las flechas indican las dependencias entre las operaciones en diferentes canales asíncronos. La aceleración obtenida con la ejecución asíncrona frente a la síncrona es de $S = 2$.

interfaz de programación (API) no expone los detalles o mecanismos específicos para el uso de operaciones asíncronas. También, hemos diseñado, implementado y probado dos backends mediante el uso de CUDA para GPUs de Nvidia, y de OpenCL para GPUs de AMD a mayores. Los modelos de ejecución síncrona y asíncrona se pueden seleccionar en tiempo de ejecución, y el solapamiento de transferencias de datos y computación es totalmente transparente.

Además, presentamos un estudio experimental con un caso que muestra la flexibilidad y sencillez del uso de nuestra propuesta. Esta experimentación refleja como las nuevas técnicas mejoran, de forma transparente, el rendimiento de las aplicaciones siempre y cuando sea posible el solapamiento. Comparamos el rendimiento y los códigos de aplicaciones de referencia programadas y ajustadas en CUDA y OpenCL. Nuestra aproximación muestra como se reduce el esfuerzo de desarrollo necesario para desarrollar un código asíncrono, consiguiendo un mayor rendimiento y portabilidad.

El resto del trabajo se organiza en las siguientes secciones. La sección II presenta el trabajo relacionado. La sección III introduce los conceptos previos sobre el modelo anterior de Controller. En la sección IV se describe el nuevo modelo asíncrono de Controller y se detalla la implementación con CUDA para su uso en dispositivos Nvidia. La sección V muestra la experimetación realizada para validar la implementación del modelo propuesto en términos de rendimiento y métricas de desarrollo de software. Por último, la sección VI concluye el trabajo, exponiendo los resultados obtenidos e indicando posibles trabajos futuros.

II. Trabajo relacionado

Existen tres principales aproximaciones para la coordinación paralela de dispositivos heterogéneos. La primera aproximación consiste en el desarrollo de soluciones a medida usando modelos paralelos nativos, como CUDA. Algunos frameworks para entornos GPGPU [5], [6] mezclan estos modelos nativos con el uso de otros modelos ajenos a los coprocesadores como OpenMP y MPI. Con esta aproximación es posible controlar de forma eficiente los recursos del hardware así como su configuración, pero el desarrollador debe tener un conocimiento avanzado de las particularidades del hardware y los diferentes modelos de programación.

Otras herramientas y librerías proponen enfoques más abstractos orientados a la paralelización automática de bucles en dispositivos heterogéneos, tales como LogFitc [7], or Maat [8]. Éstas dividen, de forma transparente, las iteraciones de un bucle en tareas mediante el uso de técnicas de transferencias asíncronas, y ejecutan las tareas en los dispositivos utilizando diferentes políticas de gestión de procesos o *scheduling*. Sin embargo, estos trabajos no tiene una coordinación entre grafos de tareas genéricas y dependencias de datos, así como las que pueden generarse en secciones con bucles anidados.

Algunos frameworks y herramientas de programación, como dCUDA [9], G-Charm [10], o los nuevos Executors propuestos para C++ [11], introducen comunicaciones asíncronas entre el host y los dispositivos GPU. Éstos solapan las tareas de comunicación y computación. En cambio, el programador necesita especificar de forma manual los mecanismo de sincronización o implementar funciones de callback para los kernels. Por lo tanto, el uso de las comunicaciones asíncronas no es transparente. Asimismo, dCUDA, Groute, y BlasX no resuelven de forma automática las dependencias de datos.

Otros sistemas de programción, tales como Gdev [12], y VAST [13], proponen abstracciones para la gestión de espacios virtuales de memoria para dispositivos GPU. Estos sistemas simplifican el acceso transversal entre el host y las GPUs. Utilizan diferentes políticas pasivas (*lazy*) o activas (*eager*) para el movimiento de datos y paginación, de forma similar a los mecanismos existentes para memoria virtual. No obstante, estos sistemas no introducen formas para hacer transparente el solapamiento de tas transferencias de estructuras de datos completas y la operaciones de ejecución de kernels tomando en cuenta las dependencias a grano grueso.

Ninguno de los enfoques mencionados resuelve el problema del solapamiento transparente de computación y transferencias de datos, para coprocesadores heterogéneos con jerarquías de memorias propias.

III. Modelo de Controllers

En esta sección se resume las características del modelo original de Controllers para plataformas heterogéneas [3]. A continuación se describen los conceptos básicos relacionados con la programación mediante el uso de este modelo, utilizando fragmentos de código ilustrativos. Se añade un ejemplo de una aplicación que consiste en una programa iterativo que aplica el filtro de Sobel a los fotogramas (*frames*) de un flujo de vídeo.

La operación de Sobel [14] procesa imágenes en escala de grises para detectar bordes. Esta aplica dos operaciones stencil a una imagen de entrada obte-

niéndo las derivadas de los colores en los ejes X e Y. La magnitud del gradiente que se computa en cada celda como la distancia Euclidiana con las correspondientes celdas restante de la matrices que se obtienen como salida del filtro. La figura 2 muestra el código con Controllers de un programa de un filtro de Sobel que envía y recibe un fotograma por iteración hacia y desde el dispositivo. En este ejemplo, el nivel de cada pixel está representado por un número real de precisión sencilla (*float*). El código de un kernel utilizado en este programa puede verse en la figura 3.

A. El modelo

El modelo original de Controller propone un instancia abstracta que coordina las actividades de ejecución de kernel y manejo de memoria en una aceleradora o coprocesador, o en un conjunto de cores de CPU. En la construcción de una instancia de Controller se le asocia a un dispositivo. Los métodos que dispone esta instancia permiten (1) asociar/desasociar estructura de datos del host al dispositivo, y (2) lanza ejecuciones de kernels usando estas estructuras de datos como parámetros.

Las operaciones de lanzamientos de kernel (*kernel launch*) introducen peticiones de ejecución dentro de una cola interna de la instancia del Controller. Los parámetros que se utilicen como entrada de la función o las estructuras de datos deben estar previamente asociadas a la instancia. El lanzamiento de un kernel también recibe un parámetro que indica el número de hilos lógicos con los que debe ejecutarse dicho kernel. El Controller maneja la granularidad necesaria, agrupando estos hilos en bloques o en tareas, para adaptarlos a las especificaciones del dispositivo. La operación de asociar (*attach*) bloquea la ejecución hasta que se ha asociado correctamente el espacio de memoria en el dispositivo y se transfieren los datos del host al dispositivo. La operación de desasociar (*detach*) recupera los datos desde el dispositivo al host y libera la memoria en el dispositivo. Esta última operación también es bloqueante, finaliza cuando todos los kernels que han sido encolados y hacen uso de los datos han terminado y los datos han sido movidos al host desde el dispositivo. Si no hay ningún kernel que tenga la estuctura de datos como salida de la función, la operación de detach simplemente desaloja la memoria del dispositivo.

B. Representación de estructura de datos: Hitmap

Hitmap [15] es una librería protable que es utilizada en el modelo de Controller para ofrecer una interfaz común para la gestión de datos en código de host y los kernels ejecutados en el dispositivo. Una estructura de *HitTile* es un *fat pointer*, un manejador para almacenar medatados además del puntero al espacio de memoria. El modelo de Controllers extiende la estructura de HitTile para almacenar metadatos relacionados con el uso de un instancia de HitTile (en adelante *tile*) en el dispositivo, incluyendo el puntero al espacio de memoria en el dispositivo. Entre las líneas 7–13 en la figura 2 muestra como se declaran

```
1  CTRL_KERNEL_PROTO( saxpy_sqrt, 1, GENERIC,
2          3,
3          IVAL, int, alpha,
4          IN, HitTile_float, matrix_x,
5          IO, HitTile_float, matrix_y );
6
7  CTRL_KERNEL( saxpy_sqrt, GENERIC,
8          int alpha,
9          KHitTile_float matrix_x,
10         KHitTile_float matrix_y,
11 {
12     const int row = threadId.x;
13     const int col = threadId.y;
14     hit(matrix_y, row, col) = sqrt(
15          alpha * hit(matrix_x, row, col) +
16                  hit(matrix_y, row, col) );
17 } );
```

Fig. 3. Ejemplo de prototipo e implementación de un kernel usando la libería Controller.

tiles con y sin memoria en el host. La línea 17 refleja como se asocian los tiles a una instancia de Controller. Los tiles sin memoria en el host se utilizan como referencias para los buffers internos que están en el dispositivo. Las operaciones de asociación de tiles a una instancia de Controller no implican movimientos de memoria, sólo el alojamiento y desalojo de memoria. La función *hit* se usa en los códigos de host y de kernel para cceder a los elementos de un tile. Esta permite un acceso portable, con un orden *row-major*, en cualquier dispositivo. Ver las líneas 14–16 en la figura 3.

C. Kernels y su implementación

En el modelo de Controller, un kernel se declara mediante dos primitivas. La primera es `CTRL_KERNEL_PROTO`, que declara el prototipo de todas las implementaciones del kernel. Ver las líneas 1–5 de la figura 3. Después del nombre del kernel, declaramos en número de las diferentes implementaciones disponibles para el proagrama. También se indica para que backends o dispositivos están dedicadas las implementaciones. Esta información es usada por el Controller para localizar en tiempo de ejecución la mejor implementación dispoible del kernel para un dispositivo en concreto. En el ejemplo hay únicamente una implementación declarada. La palabra clave `GENERIC` indica que puede ser utilizada por cualquier backend. Otras parablas clave estan asociadas a tipos específicos de backend o dispositivos. El resto del prototipo describe los parámetros del kernel, incluyendo los roles de entrada/salida, tipos y nombres. `INVAL` indica un parámetro por valor. `IN`, `OUT`, o `IO` (in/out) indica los roles que puede tomar un HitTile que se pasa por referencia.

Cada implementación de kernel se declara usando la primitiva `CTRL_KERNEL`. Ver líneas 7–17 en la figura 3. Los primero parámetros determinan el nombre, y el tipo de la implementación. Después de la declaración de los parámetros del kernel, se incluye el código de la impelemntación. El kernel del ejemplo computa la operación *saxpy*, fusiona la aplicación de las raíces cuadradas en cada uno de los elementos

```
1  int main(int argc, char *argv[]){
2      ...
3      //Controller creation
4      Ctrl ctrl = Ctrl_Create( CTRL_TYPE_CUDA, DEVICE_ID );
5
6      // Create data structures
7      HitTile_float GxC = hitTile( float, hitShape(3, 3) );
8      HitTile_float GxR = hitTile( float, hitShape(3, 3) );
9      HitTile_float imgOrig = hitTile( float, hitShape( rows, columns) );
10     HitTile_float imgGy = hitTile( float, hitShape( rows, columns) );
11     HitTile_float imgGx = hitTileNomem( float, hitShape( rows, columns) );
12     HitTile_float imgTmp1 = hitTileNomem( float, hitShape( rows, columns) );
13     HitTile_float imgTmp2 = hitTileNomem( float, hitShape( rows, columns) );
14
15     // Initialization in the host, and attachments
16     init( GxC, GxR );
17     Ctrl_Attach( GxC, GxR, imgGx, imgTmp1, imgTmp2 );
18
19     // Domain of logical threads for the device
20     Ctrl_Threads threads = Ctrl_Threads(rows, columns);
21
22     // Sobel filter: Main loop
23     // Clock: Synchronize and start measuring
24     for (int index = 0; index < ITERATIONS; index ++) {
25        getFrame( imgOrig ); // Host code to get a new frame
26        Ctrl_Attach( imgOrig ); // Attach the new image to the device
27        Ctrl_Launch(ctrl, convolutionRow, threads, imgTmp, imgTmp2, imgOrig, GxR, GxC);
28        Ctrl_Detach(ctrl, imgOrig ); // Detach to let the host reuse the tile
29        Ctrl_Launch(ctrl, convolutionColum, threads, imgGx, imgTmpGxC);
30        Ctrl_Attach(ctrl, imgGy ); // Attach the output/results buffer
31        Ctrl_Launch(ctrl, convolutionColum, threads, imgGy, imgTmpGxR);
32        Ctrl_Launch(ctrl, saxpySqrt, threads, imgGx, imgGy, 1.0);
33        Ctrl_Detach(ctrl, imgGy ); // Detach no retrieve the data
34        putFrame( imgGy ); // Host code
35     }
36     // Clock: Synchronize and stop measuring
37
38     // Free the device and host memory
39     Ctrl_Detach( ctrl, GxC, GxR, imgGx, imgTmp2, imgTmp2 );
40     Ctrl_Destroy( ctrl );
41     hit_tileFree( GxC, GxR, imgOrig, imgGy );
42     ...
43 } //main
```

Fig. 2. Fragmento de código del programa del filtro de Sobel aplicado sobre un flujo de vídeo. Está programado utilizando el modelo original de Controller.

del resultado. La implementación es una especificación genérica de paralelismo de datos a grano fino, que es adaptada por el Controller a la granularidad por tareas adecuada a los diferentes tipos de dispositivos, como GPUs, o Intel XeonPhi.

D. Codificación de programas con el modelo original de Controller

Un programa con Controllers se apoya en las siguientes pautas: (1) Creación de instancias de Controller, asociando cada una de ellas a un dispositivo que se quiera gestionar; (2) declaración, alojamiento, inicialización (si es necesario), y asociación de los tiles; (3) lanzamiento de una secuencia de kernels; (4) desasociación de los tiles para recuperar los datos en el host; y (5) la liberación de los recursos de Controller y las estructuras de datos.

La figura 2 muestra la implementación de un programa imperativo que procesa los fotogramas de un flujo de vídeo, aplicando el filtro de Sobel. Se crea una instancia de Controller en la línea 4 que utiliza el backend de CUDA, y se le asocia un dispositivo mediante su identificador. Las líneas 7–13 crean los tiles en el host, alojando la memoria en el host solo para uno de ellos que va a ser utilizado en el host. La línea 16 llama a la función de host para inicializar los tiles que contienen los pesos de las operaciones de convolución. Los tiles se asocian a la instancia del Controller en la línea 17, forzando su alojamiento en el dispositivo, y moviendo los datos de los tiles que tienen memoria en el host. La línea 20 crea una instancia que declara la lógica de indexado del espacio de los hilos a grano fino para los kernels. Entre las líneas 25 y 28 se ejecutan las operaciones de host que toman un nuevo fotograma, asocia el tile a la instancia del Controller, lanza el kernel que le procesa generando datos en los buffers internos del dispositivo y desasocia el tile para reutilizarlo en la siguiente iteración. Entre las líneas 29 y 33 se lanzan los kernels restantes. Es importante tener en cuenta que el tile que se utiliza para recuperar el resultado debe ser asociado antes de que un kernel lo utilice (línea 30), y que debe ser desasociado para recuperar estos datos (línea 33), antes de que los datos puedan ser utilizados en el host (línea 34).

El modelo original de Controllers se diseñó para aplicaciones que moviesen los datos de entrada al dispositivo al inicio del programa y retornase los resulta-

dos de vuelta al final de la computación. Por lo tanto, las operaciones para mover datos entre los espacios de memoria también gestionan el alojamiento de la memoria. Esto no es eficiente para programas con flujos de datos que necesitan transferencias de entrada y salida a los buffers en el dispositivo de forma repetida. Además, las semánticas de las operaciones de asociación de datos implican que se realizan de forma síncrona con la ejecución de los lanzamiento de los kernels antes/después, para preservar la equivalencia secuencial. Entonces, no permiten el solapamiento de computación y transferencias.

IV. El modelo asíncrono de Controller

En esta sección presentamos nuestra propuesta para el nuevo modelo asíncrono de Controllers tal que sea capa de explotar de forma eficiente y transparente las operaciones asíncronas y el solapamiento de computación y transferencia de memoria. Esta propuesta implica la redifinición de las operaciones soportadas por el Controller, la redifinición de su modelo de ejecución, y el soporte para las operaciones asíncronas en los backend implementados.

En el nuevo modelo asíncrono de Controller, tanto las tareas de kernel como las de host se identifican como unidades de computación que pueden usar y modificar los datos de un tile. Por lo tanto, las tareas de host deben ser declaradas y lanzadas de manera similar a los kernels, con prototipos que definan los roles de los parámetros, de tal manera que permita al model detectar las dependencias cuando un tile se utiliza como entrada o salida o ambas tanto en el host como en el dispositivo. Al igual que con los kernels, la ejecución de tareas de host es independiente de la ejecución del programa principal. Una vez lanzado, podrá ser ejecutado de forma concurrente a la ejecución principal por otro hilo.

En el nuevo modelo asíncrono de Controller, las operaciones de asociación de los tiles están obsoletas. Cada una es sustituida por dos operaciones, una de gestión de memoria (alojamiento y desalojo de memoria) y otra de transferencia de datos (desde y hacia el dispositivo). La transferencia de datos se puede aplicar sobre los tiles que han sido asociados previamente y tienen memoria alojada en ambos espacios de memoria del host y del dispositivo.

En esta propuesta, un *programa* para plataformas heterogéneas se ve como un código que coordina la ejecución de tareas de host y operaciones del coprocesador. Las partes del código host que procesan datos están recogidas dentro de tareas de host. El programa principal simplemente define la secuencia de peticiones que se incorporan a la cola del Controller. Esta cola está controlada por una política de ejecución. Hemos definidos dos políticas básicas: Una que ejecuta las operaciones en modo síncrono, como hacía el modelo clásico de Controlelr; y otra analiza las dependencias y ejecuta las operaciones en modo asíncrono, solapando la computación y las transferencias cuando sea posible.

A. Nuevos tipos de operaciones

Las transacciones entre el host y el dispositivo son descritas como una *secuencia de peticiones de operaciones* ($S = s_0, \ldots, s_n$) emitidas por el código principal que se ejecuta en el host. Una petición de operación (*request*) s_i puede ser uno de los siguientes tipos:

- **Allocate:** $Alloc(x)$ o $AllocDev(x)$. Petición para alojar una estructura de datos en la memoria del host o del dispositivo, o únicamente en el dispositivo, respectivamente.
- **Dealocate:** $Free(x)$. Petición para desalojar de la memoria para imágenes de una estructura de datos.
- **Host-to-Device:** $HTD(x)$. Petición para transferir los valores de la estructura de datos x desde el host al espacio de memoria del dispositivo.
- **Device-to-Host:** $DTH(x)$. Petición para transferir los valores de la estructura de datos x desde el dispositivo a la memoria del host.
- **Lanzamiento de kernel:** $K(f, In, Out)$. Petición para ejecutar un kernel en el dispositivo. Este lanzamiento recibe el nombre de la función f y dos conjuntos de estructuras de datos como parámetros. El conjunto In representa las estructuras de datos que son entradas. El otro conjunto Out contiene las referencias de las salidas. Una misma estructura de datos puede aparecer en ambos conjuntos, esto presenta el caso en el que el contenido de la estructura se lee y se escribe durante la ejecución del kernel. Los kernels también pueden tener un conjunto V de parámetros por valor. Hemos omitido este último conjunto ya que no está relacionado con las dependencias ni con con las tranferencias de datos asíncronas.
- **Lanzamiento de tareas de host:** $H(g, In, Out)$. Petición para ejecutar una función g en el host, con el mismo formato que una petición de lanzamiento de kernel, incluyendo los conjuntos de entradas y salidas.
- **Wait:** $W(x)$. Petición para bloquear la ejecución en el código principal del host hasta que todas las peticiones que involucren x hayan finalizado. En varios modelos de programación, como CUDA, esta operación está implícita por defecto después de una petición $DTH(x)$, sin embargo puede ser evitada por el programador mediante el uso explícito de transferencias asíncronas. En nuestro modelo podemos realizar una espera explícita para asegurar la portabilidad, y tener una semántica más clara en términos de sincronización. En la práctica, esta operación se utiliza únicamente cuando el código principal del programa necesita utilizar los valores del dispositivo. Por ejemplo, esta operación puede ser necesaria en un bucle con una condición de convergencia que es calculada con una operación de reducción en el dispositivo. También

```
int main(int argc, char *argv[]){
    ...
    __ctrl_block__(1)
    {
       //Controller creation
       Ctrl ctrl = Ctrl_Create( CTRL_TYPE_CUDA, DEVICE_ID );

       // Create data structures
       HitTile_float GxC = Ctrl_Alloc( ctrl, float, hitShape(3, 3) );
       HitTile_float GxR = Ctrl_Alloc( ctrl, float, hitShape(3, 3) );
       HitTile_float imgOrig = Ctrl_Alloc( ctrl, float, hitShape( rows, columns) );
       HitTile_float imgGy = Ctrl_Alloc( ctrl, float, hitShape( rows, columns) );
       HitTile_float imgGx = Ctrl_AllocDev( ctrl, float, hitShape( rows, columns) );
       HitTile_float imgTmp1 = Ctrl_AllocDev( ctrl, float, hitShape( rows, columns) );
       HitTile_float imgTmp2 = Ctrl_AllocDev( ctrl, float, hitShape( rows, columns) );

       // Domain of logical threads for the device
       Ctrl_Threads threads = Ctrl_Threads(rows, columns);

       // Initialization in the host and move to device
       Ctrl_HostTask( init, GxC, GxR );
       Ctrl_MoveTo( ctrl, GxC );
       Ctrl_MoveTo( ctrl, GxR );
       Ctrl_HostTask( getFrame, imgOrig );
       Ctrl_MoveTo(ctrl, imgOrig);

       // Sobel filter: Main loop
       // Clock: Synchronize and start measuring
       for (int index = 0; index < ITERATIONS; index ++) {
          Ctrl_Launch(ctrl, convolutionRow, threads, imgTmp, imgTmp2, imgOrig, GxR, GxC);
          Ctrl_HostTask( getFrame, imgOrig );
          Ctrl_MoveTo(ctrl, imgOrig);
          Ctrl_Launch(ctrl, convolutionColum, threads, imgGx, imgTmpGxC);
          Ctrl_Launch(ctrl, convolutionColum, threads, imgGy, imgTmpGxR);
          Ctrl_Launch(ctrl, saxpySqrt, threads, imgGx, imgGy, 1.0);
          Ctrl_MoveFrom(ctrl, imgGy);
          Ctrl_HostTask( putFrame, imgGy );
       }
       // Clock: Synchronize and stop measuring

       // Free the device and host memory
       Ctrl_Free( ctrl, GxC, GxR, imgOrig, imgGx, imgGy, imgTmp2, imgTmp2 );
       Ctrl_Destroy( ctrl );

    } //__ctrl_block__
    ...
} //main
```

Fig. 4. Fragmento del código principal del filtro de Sobel aplicado sobre un flujo de vídeo, programado utilizando la nueva implementación asíncrona de Controller.

se ha considerado una *operación global de espera* $global\_wait\_request\_W()$, sin parámetros, que bloquea la ejecución hasta que todas las peticiones previas han terminado.

B. Reglas para la ejecución asíncrona

En esta sección describimos la reglas que se aplican en tiempo de ejecución para decidir que operaciones deben esperar a estar listas para ser ejecutadas, y cuales pueden ser empezadas, realizándose en asíncrono, y pudiendo solaparse con otras operaciones previas o posteriores.

En el modelo de ejecución sínrono, todas las peticiones son ejecutadas en orden. Todas ellas se consideran operaciones bloqueantes, que paran la ejecución principal del programa hasta que terminan. La siguiente petición puede ser evaluada y preparada para evitar tiempos de inicialización, pero su ejecución no puede empezar hasta el final de la anterior.

En el modelo de ejecución asíncrono, esta regla es más flexible para permitir el solapamiento de algunas peticiones. La ejecución de kernels concurrentes es un problema ortogonal al solapamiento de transferencias y computación de kernels, por ello esta fuera del ámbito de este trabajo. Nuestra propuesta, preserva el orden secuencial de ejecución de los kernels. Estos son ejecutados uno tras otro en el mismo orden en el que se han realizado sus peticiones. De manera similar, las peticiones de ejecución de tareas de host también se realizan en orden, aunque pueden solaparse con los kernels si las dependencias lo permite. Las transferencias de datos pueden solaparse con las ejecuciones de ambas tareas de host y kernels.

La reglas que deciden cuando uan petición puede empezar se han diseñado mediante el análisis de dependencias entre los diferentes tipos de operaciones y los roles de entrada/salida de sus parámetros. Las operaciones de control de memoria (allocate/deallocate) y de espera (wait) siguen unas reglas más simples debido a su semántica. Los kernels, las tareas de host, y las transferencias de datos deben ser analizadas considerando como problemas con múltiples-

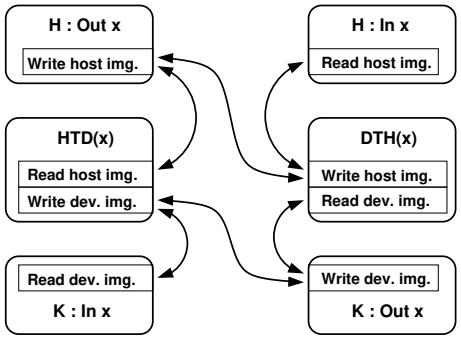


Fig. 5. Dependencias entre tipos de peticiones. Las cajas redondeadas identifican los tipos de peticiones que hay, usando x como parámetro. Se distinguen entre peticiones K o H que utilizan x con un rol de entrada o salida. Para simplificar el diagrama, no se muestran las peticiones de espera ni de gestión de memoria. Las peticiones que no están unidas por flechas se pueden ejecutar de forma concurrente.

lectores/múltiples-escritores. Cada estructura de datos puede tener dos imágenes de memoria, una en el host y otra en el dispositivo. Una operación de $HTD(x)$ lee la imagen de x en el host, y escribe en la imagen de x en el dispositivo. Una operación de $DTH(x)$ lee la imagen de x en el dispositvo, y escribe en la imagen de x en el host. Las operaciones K leen las imágenes en el dispositivo de los parámetros de entrada $x \in In$, y escriben en las imágenes en el dispositivo de los parámetros de salida $y \in Out$. Las operaciones H leen las imágenes en el host de los parámetros de entrada $x \in In$, y escriben en las imágenes en el host de los parámetros de salida $y \in Out$. Pueden haber múltiples lecturas concurrentes de una misma imágen de una estructura de datos. Varias peticiones de diferentes tipos pueden empezar si no existe una intersección entre sus parámetros, o si únicamente realizan lecturas sobre la misma imágen de los parámetros en común. Una petición r que escribe en una imagen de memoria debe esperar hasta que todas las peticiones anteriores de escritura o lectura hayan terminado. Igualmente, todas las peticiones posteriores que escriban o lean en esa imágen de memoria deben esperar por r. Este esquema de dependencias está reflejado en la figura 5.

Las reglas del modelo de ejecución asíncrona se pueden resumir de la siguiente forma:

- $Alloc(x)$ o $AllocDev(x)$: Todas las peticiones posteriores que involucren a x deben esperar hasta el final del alojamiento.
- $Free(x)$: Una petición para desalojar x debe esperar por todas las peticiones anteriores que involucren x.
- $HTD(x)$: Debe esperar si x (a) es un parámetro de una operación DTH; (b) aparece como parámetro (de entrada o salida) en un kernel anterior que no ha finalizado (operación K); (c) aparece como parámetro de salida de una tarea de host que no ha finalizado (operación H).
- $DTH(x)$: Debe espera si x (a) es un parámetro de una operación HTD; (b) aparece como parámetro de salida de un kernel que no ha finalizado (operación K); (c) aparece como parámetro (de entrada o salida) en una tarea de hots anterior que no ha finalizado (operación H).
- $K(f, I, O) : x \in I, y \in O$: Debe esperar si (a) x es un parámetro de una operación HTD sin completar; (b) y es un parámetro de una operación HTD o DTH sin completar; (c) hay una operación K previa sin completar.
- $H(g, I, O) : x \in I, y \in O$: Debe esperar si (a) x es un parámetro de una operación DTH sin completar; (b) y es un parámetro de una operación HTD o DTH sin completar; (c) hay una operación H previa sin completar.
- $W(x)$: Debe esperar por todas las operaciones previas que involucren a x.

Una petición está lista para empezar tan pronto como las condiciones de espera se hayan cumplido. Las peticiones pendientes deben ser encoladas hasta que las condiciones de disposición se cumplan. Conceptualmente, se pueden utilizar varias colas de ejecución para "ready-to-execute kernels", "ready-to-execute host codes", "ready-to-execute DTH transferences", y "ready-to-execute HTD transferences". Las operaciones de diferentes colas que estén preparadas pueden ejecutarse de forma asíncrona y pueden ser solapadas de forma segura.

C. La nueva interfaz asíncrona de Controller

Las nuevas funcionalidades definidas en el modelo asíncrono de Controller que realizan las transferencias son `Ctrl_MoveTo` para operaciones HTD, y `Ctrl_MoveFrom` para operaciones DTH. `Ctrl_Launch` se usa para invocar la computación de un kernel (operación K) como en el modelo síncrono de Controller. `Ctrl_HostTask` ha sido añadida para invocar las tareas de host (operación H). `Ctrl_Wait` implementa la operación de espera. `Ctrl_Alloc`, `Ctrl_AllocDev` y `Ctrl_Free` implementan la funciones de gestión de memoria para conjuntos de HitTiles. Las tareas de host que se quiera lanzar y ejecutar de forma asíncrona se deben declarar de forma similar a los kernels utilizando la primitiva `CTRL_HOST`.

El mismo programa de Sobel que se ha utilizado como ejemplo en la figura 2 se muestra en la figura 4 reescrita haciendo uso de la nueva interfaz del modelo asíncrono de Controller. El código de Controller se encuentra ahora contenido dentro de un bloque estructurado precedido por `__ctrl_block__()`. El parámetro indica el número de instancias de Controller que se crean y asocian a diferentes dispositivos dentro del bloque. Esta función sirve para engendrar y preparar hilos necesarios para el funcionamiento interno de cada instancia de Controller. Entre las líneas 9 y 15 se declaran y alojan las estructuras de datos con las nuevas operaciones, que también preparan la memoria del host para permitir transferencias asíncronas de memoria más eficientes (declarando la memoria como *pinned*). En el nuevo código, las operaciones de gestión de memoria únicamente aparecen al principio y al final del programa. Las transferencias del host al dispositivo y viceversa se entremez-

clan con invocaciones a tareas de host y kernels de dispositivo siguiendo el algoritmo de forma natural. Las transferencias se piden a la instancia de Controller tan pronto como las tareas de host y los kernels que generan los datos se encolan, para maximizar las oportunidades de solapamiento entre la computación y dichas transferencias.

D. Nueva política de gestión de la cola

La anterior versión del modelo de Controller define únicamente dos operaciones que involucraban trasnferencias de datos y gestión de memoria. Existen dos posibles políticas de manejo de memoria asociadas a ellas. La política *eager* mueve los datos desde/hacia el dispositivo cuando la operación se solicita. La política *lazy* mueve los datos al dispositivo exclusivamente cuando un kernel la usa por primera vez.

El modelo asíncrono de Controller introduce un rediseño de la gestión interna de la cola de peticiones. Las funciones de Controller que generan peticiones cuando son llamadas desde el código principal no son bloqueantes en el nuevo modelo, exceptuando las funciones de espera y de gestión de memoria gracias a su semántica. Todas las operaciones no bloqueantes de peticiones son encoladas en el Controller con la información sobre sus parámetros, y las referencias de las estructuras de HitTile.

La estructura de HitTile se ha expandido para almacenar información sobre su uso en operaciones de lectura y escritura para cada uno de los flujos conceptuales de las diferentes operaciones (transferencias de datos, K o H).

Hemos previsto dos módulos de políticas: Ejecución Síncrona y Asíncrona. El módulo de política síncrona simplemente bloquea la ejecución cada vez que extrae una petición y comienza su ejecución. El módulo de política asíncrona saca una petición de la cola pero se delega al backend la aplicación de las reglas presentadas en IV-B. Por lo tanto, se pueden aprovechar diferentes mecanismos de asincronía en distintos backends, utilizando las primitivas y recursos específicos por el modelo de programación correspondiente. Los métodos del backend utiliza la información dada en los parámetros y en los tiles aosciado para evaluar las reglas y transformar las peticiones en las llamadas a las funciones correspondientes del modelo de programación específico para el dispositivo.

Planteamos un sistema que traducelas dependencias generadas al aplicar las reglas descritas en IV-B a eventos o condiciones de espera, utilizando la interfaz de CUDA, para la implentación del backend del caso actual. Esto permite que el driver ejecute las peticiones en el orden adecuado de manera eficiente sin la supervisión del host.

La extensión de la estructura del HitTile contiene 6 campos, de tecnologías clásicas de concurrencia en el host y específicas de CUDA, para detectar el fin de la última operación que se realiza sobre el tile para diferentes propósitos:

- **DT_HTD:** Transferencia de datos, escritura en el dispositivo.
- **DT_DTH:** Transferencia de datos, lectura desde el dispositivo.
- **K_IN:** Parámetro de entrada en kernel, escritura en el dispositivo.
- **K_OUT:** Parámetro de entrada en kernel, lectura en el dispositivo.
- **H_IN:** Parámetro de entrada en tarea de host, escritura en el host.
- **H_OUT:** Parámetro de entrada en tarea de host, lectura en el host.

Este sistema implementa el modelo propuesto de ejecución asíncrona. Además, asegura que se respeta las ejecuciones de las peticiones en el orden correcto, siguiendo sus dependencias, y permitiendo el solapamiento de las operaciones en los diferentes streams, mediante el uso de la tecnología nativa de CUDA.

TABLA I
Resultados de las métricas de esfuerzo de desarrollo de los códigos de referencia y de Controller. Se incluye una comparativa entre el número de líneas de código (LOC), número de tokens (TOK), complejidad ciclomática de McCabe, y métrica de Halstead (Halst.).

Case study	Version	LOC	TOK	CCN	Halst.
Sobel filter	Ctrl	124	1 337	10	625 156
	CUDA_Syn	113	1 230	15	1 025 481
	CUDA_Asyn	231	1 863	24	1 265 867
	OpenCL_Syn	266	2 423	22	33 645
	OpenCL_Asyn	312	2 554	22	34 879

V. Estudio experimental

En esta sección, describimos el estudio experimental realizado para la evaluación de las ventajas y limitaciones potenciales del modelo de ejecución asíncrona en la librería de Controller. Esta sección incluye: (1) Una descripción del caso de estudio considerado; (2) un estudio de rendimiento de nuestra propuesta; y (3) una comparativa del esfuerzo de desarrollo entre la programación usando la interfaz asíncrona de Controllers o usando CUDA, u OpenCL.

A. Casos de estudio

Hemos seleccionado el programa del filtro de Sobel para probar nuestra aproximación e implementación. Este programa trabaja con valores reales de precisión simple (o *floats*). El programa aplica el filtro de Sobel a un flujo de vídeo en el dispositivo (ver figura 4). Por cada iteración se envía un fotograma al dispositivo y se retorna el resultado. Este ejemplo permite probar el solapamiento de transferencias en ambos sentidos. Para simplificar, no se tienen en cuenta las operaciones de host para las mediciones en el ejemplo para enfocar el caso en el solapamiento de las transferencias y los kernels. El programa se ha probado con imágenes Ultra-HD de tamaños de 4K, 8K y 16K.

B. Entorno de experimentación

En esta experimentación comparamos la implementación de Controller, con las implementaciones síncrona y asíncrona de programas de referencia. El código de Controllers es el mismo para ambos modos de ejecución. Ambas políticas de ejecución se puede seleccionar en tiempo de ejecución.

Los experimentos se han ejecutado sobre una plataforma equipada con una GPU de Nvidia. La máquina anfitriona se compone de un procesador Intel i5-3330 @3.0GHz, con 8 GB GDDR3 de memoria principal, y esta equipada con una NVIDIA's Tesla K40c, con 2880 cores @ 745 MHz, y 12 GB GDDR4 de memoria global. El S.O. es una distribución Linux CentOS 7. Todos los programas se han compilado con GCC v7.2, utilizando el toolkit y el driver de CUDA 10.0, y OpenCL 1.2. Por simplicidad, los tamaños de bloques usados en la ejecuciones de los kernels de GPU son de 16×16. La experimentación de rendimiento mide el tiempo de reloj desde el comienzo hasta el final del bucle principal del programa. Este tiempo incluye las transferencias de datos, la computación, y los sobrecostes del sistema.

C. Estudio de rendimiento

Los tiempos de ejecución del programa del filtro de Sobel se pueden observar en la tabla II. El programa del filtro de Sobel no mide las operaciones de host, y se ha enfocado en las transferencias de datos en ambas direcciones. La mejoría observada proviene del solapamiento de los kernels, las transferencias de datos en ambas direcciones. En la figura 6 se muestra gráficamente la representación de la ejecución que se obtiene del profiler visual de CUDA de la ejecución de ambas versiones de Controllers. El solapamiento de las transferencias de datos en ambas direcciones provoca que tarden un poco más de tiempo, pero permite un mayor solapamiento de ambas transferencias y los kernels. Este hecho lleva a altas mejoras en el rendimiento del modo asíncrono.

Comparando los resultados de Controller con las correspondientes referencias (síncrona o asíncrona), las mediciones indican un pequeña y estable penalización de tiempo, debido a la gestión del sistema de cola y los mecanismos de sincronización de Controller.

D. Métricas de esfuerzo de desarrollo

Esta sección analiza las diferencias en el esfuerzo de desarrollo entre los códigos de Controller y las implementaciones base utilizando CUDA para los escenario asíncronos. Hemos medido las siguientes métricas de esfuerzo de desarrollo: Número de líneas de código; Número de tokens; Complejidad Ciclomática de McCabe [16] y la métrica de desarrollo de Halstead [17]. Las primeras métricas midel el volumen de código que el programador debe desarrollar. La tercera cuantifica de forma racional el esfuerzo necesario para programar en términos de divergencias de código ye incidencias potenciales que pueden ser consderadas en el desarrollo, prueba y debug del programa. La última métrica se basa en la complejidad y volumen de código para obtener una medición del esfuerzo de desarrollo. Los códigos empleados incluyen las definiciones de los kernels, las caracterizaciones de los kernels, el código principal del host, y las estructuras de datos empleadas. Hay un importante volumen de código dentro de los kernels, y es igual para todas las versiones. Los códigos de tareas de host del filtro de Sobel de han excluido del estudio.

Los resultados mostrados en la tabla I indican que la programación usando Controllers genera un menor volumen de código, y reduce las métricas de Halstead. Esto es se puede observar especialmente en que las versiones base asíncronas introducen manualmente mencanismos más complejos de sincronización. Estos mecanismos son transparentes en los programas de Controllers. Un análisis más en detalle revela una alta reducción de las métricas en las partes de código del host correspondientes al flujo principal del programa, como se esperaba. Estos resultados indican un menor esfuerzo de desarrollo necesario en lo programas de Controllers en comparación de usar CUDA u OpenCL directamente.

VI. Conclusiones

Este trabajo persenta una técnica para detectar y ejecutar de forma transparente y asíncrona transferencias de datos entre una máquina anfitriona y un acelerador de harware. La implementación extiende las funcionalidades de Controller, rediseñando la estructura interna, e incluyendo una nueva gestión del sistema de colas con políticas que soportan modos de ejecución síncrona y asíncrona. Este modo se puede elegir en tiempo de ejecución. Se han desarrollado dos backends, uno utilizando CUDA, y otro con OpenCL, que demuestran como esta técnica puede ser aplicada sobre un amplio rango de dispositivos.

La implementación propuesta solapa automáticamente las transferencias de memoria con la computación de kernesl y la ejecución de código de host, sin añadir una complejidad significativa. El estudio experimental muestra mejoras notables en el rendimiento de programas de filtro sobre los fotogramas de un vídeo con transferencias intensivas en los ambos sentidos. Las métricas de desarrollos indican que los códigos de Controllers son más sencillos de desarrollar que una implementación manual con las librerías y herramientas propias de los dipositivos utilizados.

Como trabajo futuro se incluye el desarrollo de nuevos backends para soportar otros tipos de dispositivos con diferentes restricciones y otras políticas de gestión de colas más sofisticadas.

Referencias

[1] TOP500.org, "Top500 Supercomput. Sites," Dec 2017, on `http://www.top500.org`.

[2] Anthony Danalis, Lori Pollock, Martin Swany, and John Cavazos, "Mpi-aware compiler optimizations for improving communication-computation overlap," in *23rd Int. Conf. on Supercomput.*, New York, NY, USA, 2009, ICS '09, pp. 316–325, ACM.

[3] Ana Moreton–Fernandez, Hector Ortega–Arranz, and Arturo Gonzalez–Escribano, "Controllers: An abstraction

TABLA II

TIEMPOS DE EJECUCIÓN DEL PROGRAMA DEL FILTRO DE SOBEL. LA ÚLTIMA COLUMNA MUESTRA EL PORCENTAJE DE MEJORA DE RENDIMIENTO ENTRE EL MODOS ASÍNCRONO SOBRE EL MODO SÍNCRONO DE CONTROLLER.

Sobel filter (CUDA-Backend on NVIDIA GPU)

Input Size	CUDA-Syn	CUDA-Asyn	Ctrl-Syn	Ctrl-Asyn	% Improvement
4 096 × 2 160	0,975	0,586	0,984	0,592	39,8
7680 × 4 320	3,734	2,140	3,760	2,154	42,7
15 360 × 8 640	14,950	8,396	15,047	8,452	43,8

Sobel filter (OpenCL-Backend on NVIDIA GPU)

Input Size	OpenCL-Syn	OpenCL-Asyn	Ctrl-Syn	Ctrl-Asyn	% Improvement
4 096 × 2 160	1,222	0,675	1,228	0,680	44,6
7680 × 4 320	4,596	2,527	4,584	2,541	44,5
15 360 × 8 640	18,382	10,202	18,276	10,152	44,4

Sobel filter (OpenCL-Backend on AMD GPU)

Input Size	OpenCL-Syn	OpenCL-Asyn	Ctrl-Syn	Ctrl-Asyn	% Improvement
4 096 × 2 160	0,597	0,595	1,122	0,705	37,1
7680 × 4 320	2,166	2,127	3,377	2,270	32,6
15 360 × 8 640	8,718	8,406	14,786	8,702	40,8

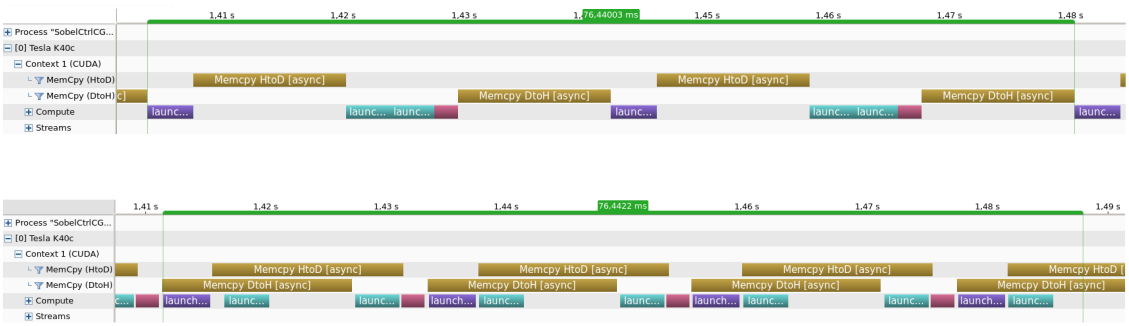


Fig. 6. Capturas de pantalla de la herramienta CUDA 10.0 Visual Profiler mostrando 76ms. de ejecución del programa de Controller del filtro de Sobel. Arriba: modo de ejecución síncrono. Abajo: modo de ejecución asíncrono con solapamientos entre transferencias, kernels y tareas de host.

to ease the use of hardware accelerators," *The International Journal of High Performance Computing Applications*, vol. 0, no. 0, pp. 16, 2017.

[4] Ana Moreton-Fernandez, Eduardo Rodriguez-Gutiez, Arturo Gonzalez-Escribano, and Diego R. Llanos, "Supporting the Xeon Phi coprocessor in a heterogeneous programming model," in *Euro-Par 2017: 23rd International Conference on Parallel and Distributed Computing, Santiago de Compostela, Spain, August 28 – September 1, 2017, Proceedings*, Santiago de Compostela, Galicia, Spain, 2017, pp. 457–469, Springer International Publishing.

[5] M. Sourouri, J. Langguth, F. Spiga, S. B. Baden, and X. Cai, "Cpu+gpu programming of stencil computations for resource-efficient use of gpu clusters," in *18th Int. Conf. on Computational Science and Engineering*, Porto, Portugal, Oct 2015, pp. 17–26, IEEE.

[6] Stephen F. Siegel, Manchun Zheng, Ziqing Luo, Timothy K. Zirkel, Andre V. Marianiello, John G. Edenhofner, Matthew B. Dwyer, and Michael S. Rogers, "Civl: The concurrency intermediate verification language," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, New York, NY, USA, 2015, SC '15, pp. 61:1–61:12, ACM.

[7] Antonio Vilches, Angeles Navarro, Francisco Corbera, Andres Rodriguez, and Rafael Asenjo, "heterogeneous parallel for template based on tbbs," in *Proceedings of the 10th International Symposium on High-Level Parallel Programming and Applications*, Valladolid, Spain, 2017, Springer International Publishing.

[8] Borja Pérez, José Luis Bosque, and Ramón Beivide, "Simplifying programming and load balancing of data parallel applications on heterogeneous systems," in *Proceedings of the 9th Annual Workshop on General Purpose Processing Using Graphics Processing Unit*, New York, NY, USA, 2016, GPGPU '16, pp. 42–51, ACM.

[9] T. Gysi, J. Bär, and T. Hoefler, "dcuda: Hardware supported overlap of computation and communication," in *SC16: International Conference for High Performance Computing, Networking, Storage and Analysis*, Salt Lake City, Utah, EE. UU., Nov 2016, pp. 609–620, IEEE.

[10] R. Vasudevan, Sathish S. Vadhiyar, and Laxmikant V. Kalé, "G-charm: An adaptive runtime system for message-driven parallel applications on hybrid systems," in *ICS 2013 - Proceedings of the 2013 ACM International Conference on Supercomputing*, Eugene, Oregon, United States, 7 2013, pp. 349–358, ACM.

[11] C++ Executors, "C++ Standards Committee Papers," 2018, `goo.gl/bf4evL`,Last visit: August, 2018.

[12] Shinpei Kato, Michael McThrow, Carlos Maltzahn, and Scott Brandt, "Gdev: First-class gpu resource management in the operating system," in *Proceedings of the 2012 USENIX Conference on Annual Technical Conference*, Berkeley, CA, USA, 2012, USENIX ATC'12, pp. 37–37, USENIX Association.

[13] Janghaeng Lee, Mehrzad Samadi, and Scott Mahlke, "VAST: the illusion of a large memory space for GPUs," in *Proceedings of the 23rd International Conference on Parallel Architectures and Compilation*, New York, NY, USA, 2014, PACT '14, pp. 443–454, ACM.

[14] Rafael C. Gonzalez and Richard E. Woods, *Digital Image Processing (3rd Edition)*, Prentice Hall, New Jersey 07458, 3 edition, Aug. 2007.

[15] Arturo Gonzalez-Escribano, Yuri Torres, Javier Fresno, and Diego R. Llanos, "An Extensible System for Multilevel Automatic Data Partition and Mapping," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 5, pp. 1145–1154, 2014.

[16] Thomas J McCabe, "A complexity measure," *Software Engineering, IEEE Transactions on*, vol. 4, pp. 308–320, 1976.

[17] Maurice H Halstead, *Elements of Software Science (Operating and programming systems series)*, Elsevier Science Inc., New York, NY, USA, 1977.

Mecanismo de equilibrado de carga en sistemas heterogéneos

Fernando Alonso [1] , Arturo Gonzalez-Escribano[2], Yuri Torres[2] y Diego R. Llanos[2]

Resumen— **El reparto de la carga de trabajo en sistemas heterogéneos es una tarea complicada ya que no todos los nodos de un sistema contienen los mismos recursos computacionales. El tipo de distribuciones de datos más comúnmente utilizado es el reparto equitativo entre todos los procesos. Sin embargo, para los sistemas heterogéneos es necesario una política de reparto más sofisticada. Este trabajo propone la integración de un sistema de reparto de la carga computacional por pesos en una herramienta programación paralela distribuida. Para utilizarlo se indica el porcentaje de carga total que se desea otorgar a cada proceso. De esta forma, se puede explotar al máximo la capacidad de cómputo de cada uno de los dispositivos que componen el sistema. Nuestro estudio experimental muestra que nuestra propuesta se adapta perfectamente al contexto del modelo de programación escogido, consiguiendo una mejora de rendimiento significativa comparado con una distribución equitativa de la carga de trabajo.**

Palabras clave— **Balanceo de Carga, Computación Heterogénea, Distribución de la Carga, HPC, Particionado de Datos.**

I. Introducción

Actualmente, los sistemas heterogéneos proporcionan una gran capacidad computacional y bajo coste energético ya que son capaces de explotar, de forma conjunta, dispositivos de naturaleza diferente, tales como CPUs, GPUs, FPGAs [1, 2]. La programación de este tipo de sistemas requiere un conocimiento detallado de los recursos hardware que componen el sistema, tanto para gestionar y explotar cada dispositivo como para asignar la cantidad de carga computacional a cada uno de ellos.

Realizar una distribución correcta de la carga en un sistema heterogéneo no es una tarea sencilla. En general, las técnicas de particionado y mapeado de datos más comunes, se apoyan habitualmente, en una distribución equitativa de los datos por nodo. En un sistema heterogéneo, un particionado de datos equitativo no aprovecha, de forma eficiente, toda la capacidad de cómputo debido a la diferencia de recursos hardware de cada uno de las unidades de cómputo.

Hitmap [3] es una librería desarrollada por el Grupo de Investigación Trasgo [4], pensada para el particionado de datos (tiling) jerárquico y el mapeado de arrays y estructuras dispersas. Hitmap propone un array multidimensional a través de un nuevo tipo de dato, el Tile. Hitmap realiza una distribución de Tiles o SubTiles automática entre los diferentes nodos de un sistema ya que enmascara todas las operaciones explícitas de particionado, reparto y comunicación de datos a través de una interfaz, sencilla de utilizar, en lenguaje C. Aunque se puede elegir entre múltiples tipos de particionado, a través de los Layouts (bloques, cíclica, por dimensión, particionado para estructuras dispersas, etc. . .). Sin embargo, no es posible realizar una distribución de datos en los que se puede elija la carga o el volumen de datos que se le otorga a cada nodo.

Este trabajo propone el desarrollo de un plug-in para la herramienta Hitmap, que permita realizar un particionado de datos por pesos para distribuir la carga computacional proporcional a la capacidad computacional de cada uno de los nodos de un sistema heterogéneo. El plug-in permite un reparto en datos de una y varias dimensiones así como la elección el peso de los datos por nodo y por proceso.

El trabajo está organizado en las siguientes secciones: En la sección II se muestran los trabajos relacionados. La sección III describe la librería Hitmap. La sección IV ropuesta del trabajo. La sección V describe los detalles de la implementación. La sección VI contiene los resultados experimentales. Finalmente, la sección VII presenta las conclusiones obtenidas y el trabajo futuro.

II. Trabajo relacionado

Aumentar la localidad de los datos respectos a los elementos de cómputo que los utilizan permite reducir los costes de comunicaciones o de acceso a los datos. Las técnicas orientadas a ello se han convertido en un elemento clave para la paralelización efectiva de aplicaciones. Especialmente en el contexto de los actuales sistemas paralelos, cada vez más heterogeneos, más complejos en sus jerarquías de memoria, con espacios de datos separados y/o distribuidos [5].

En modelos de programación clásicos, como HPF (High-Performance-Fortran) [6, 7] ya se considera la idea de expresar afinidades entre elementos de estructuras de datos y de proceso, o elementos de estructuras de datos que deben ser mapeadas en el mismo elemento de proceso. La mayor parte de las técnicas propuestas en este sentido se han orientado a un análisis en tiempo de compilación.

Un objetivo perseguido por diversas propuestas de lenguajes o modelos de programación paralela ha sido introducir abstracciones que separen las especificaciones de los algoritmos de las técnicas de partición y distribución de las estructuras de datos. De esta forma, el programador puede probar diversas estrategias de partición sin necesidad de reorganizar el código por completo. Esto es muy evidente, por ejemplo, en los modelos de programación de tipo PGAS, como Chapel [8], STAPL [9] o DASH [10].

[1]Univ. Valladolid, e-mail: fernando.alonso@alumnos.uva.es
[2]Dpto. de Informática, Univ. Valladolid, e-mail: {yuri.torres|arturo|diego}@infor.uva.es

En cuanto a las técnicas de partición y distribución especificamente orientadas a sistemas heterogéneos, ya en [11], se muestran algunas técnicas que permiten realizar una asignación de carga a cada nodo de una forma eficiente, aunque no se diseñe una política o mecanismo de particionado. En el año 2005, Dongarra et.al [12] se focalizan en la computación heterogénea y los clústers, describiendo el trabajo futuro por hacer. En [13] se realiza un estudio del estado del arte sobre la computación heterogénea, centrándose en el uso de algunos co-procesadores como GPUs, FPGAs junto con los procesadores tradicionales para la resolución de problemas en sistemas heterogéneos. Más modernamente, el trabajo de Unat et.al [5] clasifica las técnicas actuales sobre localidad de datos en general, mostrando las tendencias actuales a utilizar sistemas dinámicos de equilibrado de carga, pero basados en tareas de grano fino o medio, con el consiguiente sobrecoste en el sistema de ejecución.

Nuestra propuesta se presenta como una técnica sencilla de reparto de datos en grano grueso, apropiado para sistemas distribuidos o con alto coste de transferencia de datos. Se integra de forma modular dentro de un sistema de programación paralela, sin necesidad de reestructurar el código para utilizarla. Se aplica en tiempo de ejecución, permitiendo adaptar la computación a la plataforma sin necesidad de recompilar el código.

III. Librería Hitmap

En esta sección se introducirán algunos conceptos sobre la librería Hitmap [3], tales como la arquitectura y Layouts de la misma.

A. Conceptos clave

Hitmap es una librería altamente eficiente pensada para particionado de datos (tiling) jerárquico y mapeado de arrays y estructuras dispersas. [14] Hitmap introduce los siguiente conceptos:

Signature: Una *signature* S se define como una terna que representa un subespacio de índices sobre un array unidimensional. La cardinalidad de una signature es el número de índices diferentes del dominio.

$$S \in \text{Signature} = (\text{begin} : \text{end} : \text{stride})$$
$$\text{Card}(S) = \lfloor (s.end - s.begin + 1)/s.stride \rfloor$$

Shape: Un *shape* h es una n-pla de signaturas. Representa una selección de un subespacio de los índices de un array con dominio multidimensional. La cardinalidad de una shape es el número de diferentes combinaciones de índices del dominio.

$$h \in \text{Shape} = (S_0, S_1, S_2, \ldots, S_{n-1})$$
$$\text{Card}(h \in \text{Shape}) = \prod_{i=0}^{n-1} \text{Card}(S_i)$$

Tile: Un *tile* es un array n-dimensional. Su dominio es definido por un shape, y tiene un tiene un número de elementos de un tipo (*type*) dado.

$$\text{Tile}_{h \in \text{Shape}} : (S_0 \times S_1 \times S_2 \times \ldots S_{n-1}) \rightarrow \texttt{<type>}$$

B. Arquitectura

La figura 1 muestra la arquitectura de la librería Hitmap.

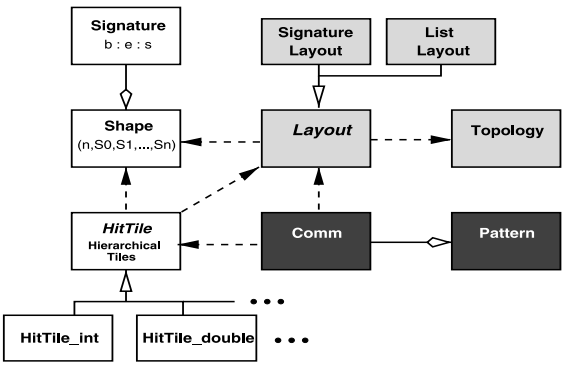


Fig. 1
Arquitectura de Hitmap

Existen tres conjuntos de funcionalidades distinguibles en Hitmap:

- ***Tiling:*** Parte encargada de la definición y manipulación de arrays y tiles. Los HitTiles, que podemos clasificar de forma general como arrays multidimensionales, son la estructura de datos principal en la que se fundamenta Hitmap. Los HitTiles pueden ser de cualquier tipo básico e incluso de nuevos tipos definidos por el usuario y están definidos por un Shape. Dichos Shapes, incluyen una Signature por cada dimensión que tiene el Tile. De esta forma, un HitTile, bidimensional que contuviese, por ejemplo, 10 posiciones en cada dimensión, contaría con un Shape, de dos elementos, que llevaría la asignatura $0 : 9 : 1$ asociada tanto en S_0 con en S_1.

- ***Mapeo:*** Estas funciones se encargan de la distribución y disposición de datos. A través de la topología (*Topology*), se elige una una distribución de los procesadores determinada. Algunos ejemplos de Topologies son: plana (1 dimensión), rectangular o cuadrada (2 dimensiones). De esta forma, los procesos, pueden intercambiar datos en cualquiera de las direcciones que permita la Topology. Los Layouts, en cambio, nos permiten elegir una distribución de la carga determinada, según el propio Layout, a través de los procesos y de forma automática. Para poder utilizar un Layout sobre un HitTile, se necesita establecer una Topology. Mediante los Layouts, los procesos pueden conocer que parte del Tile es la suya. Además, existen varios Layouts diferentes, dependiendo de como se quiera repartir la carga. Algunos ejemplos son: plug_lay_Blocks, que

reparte el Tile por bloques de forma equitativa entre todos los procesadores o plug_lay_Cyclic, que reparte el Tile de forma cíclica y equitativa entre todos los procesos.

- ***Comunicación:*** Por último, la librería también incluye toda una parte de comunicaciones. Mediante la información que proporciona el Layout y la Topology (incluida en el Layout), las funciones de comunicación de Hitmap permiten el intercambio de Tiles o elementos de Tiles entre procesos, de una forma sencilla. Además, en HitMap, existen los Communication Patterns, que permiten establecer un patrón o secuencia de comunicaciones básicas de forma automática. El Comm. Pattern, almacena la secuencia de comunicaciones que se desea realizar, y, simplemente, permite ejecutar dicha secuencia de comunicaciones en un punto del programa, de una forma sencilla.

IV. Propuesta

Estas sección presenta la propuesta de equilibrado de carga para sistemas heterogéneos.

A. Modelo propuesto

Hitmap dispone de varios tipos de distribuciones de carga, a través de los Layouts. Sin embargo, no incluye ninguna forma de reparto en la que se pueda especificar distintos tamaños de carga para cada proceso.

El propósito de este trabajo es dotar a Hitmap de un reparto de carga por bloques balanceado según el peso que se le desee otorgar a cada proceso. De esta forma, se puede equilibrar la carga computacional que tiene cada proceso, indicándole el peso que queremos que tenga respecto a la estructura de datos original. Dicho reparto de carga puede realizarse respecto a una dimensión del Tile. Para estructuras unidimensionales, simplemente, se le asigna a cada proceso una parte del Tile que corresponda con el peso otorgado. En la figura 2 se muestra un ejemplo de la distribución de la carga por pesos en un Tile de 10 elementos $(0:9:1)$ de una sola dimensión y tres procesos distintos. A Proc 1 se le otorga un peso de 0.5 del Tile, a Proc 2 se le proporcionó un peso de 0.2 y a Proc 3 0.3.

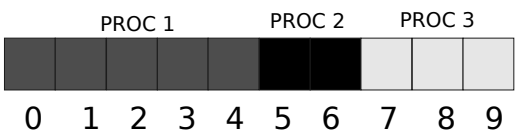


Fig. 2
Tile distribuido por pesos entre tres procesadores

Para los HitTiles multidimensionales, se permite repartir por pesos en una sola dimensión en cada particionado, es decir, repartimos sola la signatura de una dimensión por pesos manteniendo las otras signaturas de las dimensiones restantes igual que en el HitTile original. Por ejemplo, dado un Tile T, con un Shape $S/S \in \text{Shape} = (0:9:1, 0:9:1, 0:9:1)$, si se quiere repartir la primera dimensión (S_0) en $[0.5, 0.2, 0.3]$, entre los procesadores 1,2, y 3, como se ha hecho en el caso de un Tile unidemensional, los subtiles que obtendrá cada proceso, tendrían los siguientes shapes:

$$\textit{Proceso 1: } S_{P1} = (0:4:1, 0:9:1, 0:9:1)$$
$$\textit{Proceso 2: } S_{P2} = (5:6:1, 0:9:1, 0:9:1)$$
$$\textit{Proceso 3: } S_{P3} = (7:9:1, 0:9:1, 0:9:1)$$

De esta manera, se permite repartir por filas o columnas en una estructura bidimensional. Para repartir un HitTile multidimensional por varias dimensiones con diferentes pesos en cada dimensión, se puede aplicar la partición n veces, una por cada dimensión, y, conseguir una sistema de reparto por pesos jerárquico. En la figura 3, se muestra una matriz o Tile de dos dimensiones, distribuido entre tres nodos por filas y dentro de cada nodo se ha vuelve a aplicar la partición en cada respectivo subtile por columnas. De esta manera, se pueden realizar repartos por pesos por nodos y dentro de cada nodo de los nodos un reparto por pesos entre los procesos de dicho nodo.

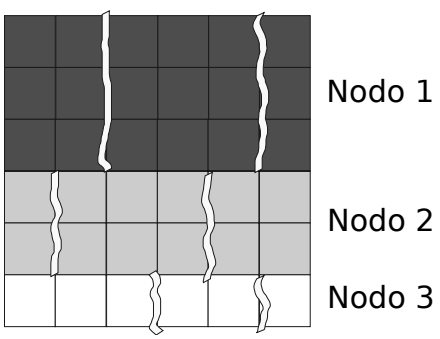


Fig. 3
Tile multidimensional distribuido por pesos

La forma correcta de agregar nuevos tipos de particionado de datos en Hitmap, es mediante la implementación de nuevos Layouts. Con la creación de un nuevo Layout, conseguimos integrar en la librería Hitmap un nuevo tipo de reparto de carga. En este caso, se realizarán dos nuevos Layouts. El primero y principal, permitirá el reparto por bloques, dados unos determinados pesos y una determinada dimensión, y operará como acabamos de explicar en los párrafos anteriores. También, implementaremos otro Layout que nos permitirá un reparto por pesos solamente en una dimensión, y, las dimensiones restantes repartirá con Layout de reparto de bloques simple, la carga. Profundizaremos en los detalles de ambos, en la sección siguiente.

Ambos Layouts, poseerán un parámetro que será, un array de números reales (float), en el que se indicarán los ratios de carga deseados para cada pro-

ceso. La posición en el array, indicará el número de proceso al que va destinado el peso almacenado. Para distribuir la carga entre nodos, debemos hacerlo de forma manual, es decir, comprobando que procesadores pertenecen a cada nodo y asignándoles un peso. De esta forma, se puede otorgar a los procesos de los nodos más lentos de un sistema heterogéneo, una carga computacional menor, y, por el contrario, a los nodos más rápidos un mayor ratio del HitTile, permitiendo una mejor explotación del sistema y sus recursos.

B. Integración con Hitmap

Se desarrollar un sistema para el balanceo de carga en sistemas heterogéneos mediante la adición de un plug-in a la librería Hitmap debido a las razones siguientes:

En primer lugar, la librería Hitmap ya posee toda la funcionalidad necesaria para el particionado de datos jerárquico y el mapeado de arrays, es decir, tenemos una librería que nos provee la funcionalidad de particionado, reparto de carga, comunicaciones, etc... de forma transparente al usuario. Los Layouts de Hitmap están diseñados para poder ampliarse sin necesidad de modificar otras partes de la librería reduciendo así, la tarea de programación.

En segundo lugar, se pueden ejecutar todas las aplicaciones que sean desarrolladas con el nuevo sistema de balanceo de carga con la librería Hitmap únicamente seleccionando el Layout específico y deseado para dicho código. Como se muestra en la figura 4, declarar un Layout en Hitmap es tan sencillo como llamar a *hit_layout*, con los argumentos que necesite el Layout. Es necesario cambiar la declaración de la primera línea por la declaración de la segunda (figura 4).

```
1 HitLayout oneLayout = hit_layout(
      ↪ plug_layBlocks, topo, shape );
2 HitLayout otherLayout = hit_layout(
      ↪ plug_layBlocksWeighted, topo, shape,
      ↪ dim, weights);
```

Fig. 4
Ejemplo de declaración de Layouts

Por último, el HitTile, de la librería Hitmap se utiliza como estructura de datos base en la librería Controller [15], también desarrollada por el Grupo de Investigación Trasgo. Por ello, la distribución balanceada de carga se puede utilizar también en Controller sin tener que desarrollar específicamente el software necesario para implantarla en la librería.

V. Detalles de Implementación

En esta sección se muestra la implementación final del plug-in para Hitmap. Dicho plug-in consiste en el desarrollo de dos Layouts que reparten la carga por pesos entre procesos.

A. Layout plug_layDimBlocksWeighted

Mediante el uso de este Layout se realiza un reparto de la carga por bloques siguiendo los pesos indicados para cada proceso en una sola dimensión. Se deja el resto de las dimensiones completas para cada proceso. Este tipo de partición queda descrito en sección IV-A como modelo principal de reparto por pesos.

```
1 /* plug_layDimBlocksWeighted */
2 typedef struct
3 {
4    int num_procs;
5    float *ratios;
6 } Load_ratios;
7
8 HitLayout hit_layout( plug_layDimBlocksWeighted
      ↪ , HitTopology topo, HitShape shape, int
      ↪  restrictDim, Load_ratios weights);
```

Fig. 5
Interfaz del Layout plug_layDimBlocksWeighted

En la figura 5 se muestra la interfaz del Layout. A través de Load_ratios, se le indica el número de procesos totales, junto con el peso de cada proceso. Cada posición del array, indica el número de proceso, y, el contenido de la posición, el peso que se le da a dicho proceso. El parámetro restrictDim, indica la dimensión a la que se va a aplicar el Layout por pesos. Dichos pesos, no han de sumar 1 necesariamente. El ratio de carga de cada procesador será siempre $\frac{weights[proc]}{\sum_{i=0}^{procs-1} weights[i]}$, por lo que, se puede usar la escala que se desee. En cuanto al número de procesos, solo se activarán aquellos que se pase como parámetro a Load_ratios, es decir, si se pasamos 4 procesos y el programa se lanza con 8 procesos, estarán activados $[0, 3]$ y desactivados $[4, 7]$. Si, por el contrario, el número de procesos que se le pasa al Layout es mayor que el número de procesos con los que ha lanzado el programa, se ignorarán los pesos que no pertenezcan a ningún proceso real.

B. Layout plug_layBlocksWeightedToSelectedDim

Por otra lado, este segundo Layout, realiza la repartición de la dimensión seleccionada por pesos, y las demás dimensiones las reparte por bloques de forma equitativa. Está pensado para realizar repartos sencillos y rápidos en Topologies 2D. De esta forma, una de las dimensiones se parte por pesos y la otra u otras equitativamente.

Por ejemplo, si tenemos una Topology 2D de 4×2 procesadores y, un HitTile de 10×10 elementos y, seleccionamos la dimensión 0 (filas) para el reparto balanceado, con los pesos $[0.3, 0.3, 0.2, 0.2]$ para cada proceso, obtendremos los siguientes elementos:

$$\begin{aligned}
&Procesos\ [P_{00}, P_{01}] \rightarrow 0.3 * 1/2 * 100 = 15\\
&Procesos\ [P_{10}, P_{11}] \rightarrow 0.3 * 1/2 * 100 = 15\\
&Procesos\ [P_{20}, P_{21}] \rightarrow 0.2 * 1/2 * 100 = 10\\
&Procesos\ [P_{30}, P_{31}] \rightarrow 0.2 * 1/2 * 100 = 10
\end{aligned}$$

En laTabla I podemos ver el Shape exacto que obtendría cada proceso, en el ejemplo:

TABLA I
SHAPE DE CADA PROCESO DE UNA TOPOLOGY 2D DE 4×2 PROCESADORES Y, UN HITTILE DE 10×10 ELEMENTOS

	0	1
0	(0:2:1,0:4:1)	(0:2:1,5:9:1)
1	(3:5:1,0:4:1)	(3:5:1,5:9:1)
2	(6:7:1,0:4:1)	(6:7:1,5:9:1)
3	(8:9:1,0:4:1)	(8:9:1,5:9:1)

En cuanto a la interfaz de este Layout, es similar a la anterior, y los campos necesarios también son iguales. A través de Load_ratios, se le indica el número de procesos totales, junto con el peso de cada proceso en la dimensión seleccionada. El parámetro selectedDimToWeight, indica la dimensión a la que se va a aplicar el Layout por pesos. Dicha interfaz se muestra en la figura 6

```
1 /* plug_layDimBlocksWeighted */
2 typedef struct
3 {
4 int num_procs;
5 float *ratios;
6 } Load_ratios;
7
8 HitLayout hit_layout(
      ↪ plug_layBlocksWeightedToSelectedDim,
      ↪ HitTopology topo, HitShape shape, int
      ↪ selectedDimToWeight, Load_ratios
      ↪ weights);
```

Fig. 6
INTERFAZ DEL LAYOUT PLUG_LAYBLOCKSWEIGHTEDTOSELECTEDDIM

VI. ESTUDIO EXPERIMENTAL

En esta sección se realizará un estudio experimental para comprobar la aceleración obtenida en un cluster heterogéneo, utilizando un reparto de carga balanceado según la capacidad computacional del nodo.

Para realizar la experimentación, vamos a emplear tres máquinas diferentes del cluster del Grupo Trago, que es heterogéneo. Utilizaremos 12 procesos en cada una, ya que, es el máximo de una de estas máquinas, y, para esta experimentación, usaremos el mismo número de procesadores en todos los nodos (en este caso 3). Estas máquinas son Manticore, Heracles e Hydra. Sus especificaciones son las siguientes:

Maticore:
- CPU: 2 x Intel Xenon Platinum 8160 @ 2.10GHz (96 cores), memoria: 256 GB DDR4 y coprocesaores: NVIDIA TESLA V100.

Heracles:
- CPU: AMD Opteron 6376 @ 2.3 GHz (64 cores) y memoria: 250 GB.

Hydra:
- CPU: 2 x Xenon E5-2690v3 @ 1.9 GHz (12 cores) memoria: 64GB y coprocesaores: 4 x NVIDIA GTX TITAN BLACK 2880.

En orden decreciente, Manticore, Heracles e Hydra son las máquinas con más recursos computacionales. Otorgaremos más carga computacional a los nodos con mejor capacidad de procesamiento. Por ello, la distribución de carga será la siguiente:

15 % de la carga se dispondrá en Hydra.
38 % de la carga se ubicará en Heracles.
47 % de la carga se situará en Manticore.

En cada máquina, los 12 procesos tendrán el mismo ratio de carga.

Para verificar la velocidad de ejecución en diferentes entradas se utilizara la aplicación Stencil de Jacobi en 2D [16], que ya ha sido analizado, resuelto y probado en Hitmap. Mostraremos la mejora en tiempo de ejecución al utilizar una distribución en bloques por pesos frente a usar un reparto equitativo por bloques entre los tres nodos elegidos del cluster. Utilizaremos, por una parte, el Layout mostrado en la sección V-A, para medir el tiempo de ejecución cuando se reparte la matriz por filas de forma balanceada, según la carga indicada, entre todos los procesos. Por otro lado, utilizaremos *lay_DimBlocks* como Layout de reparto por bloques y por dimensión de forma equitativa, para medir el tiempo de ejecución de un reparto por filas de la matriz entre todos los procesos.

A. Resultados

Medimos el tiempo de ejecución empleado por el ejemplo del Stencil de Jacobi en 2D, con ambos Layouts (equitativo y balanceado por pesos). Por una parte, a medida que aumentan las iteraciones, con un tamaño de matriz fijo figura 7. Por otro lado, con un número de iteraciones fijo (1000), a medida que aumenta el tamaño de la matriz.

En la figura 7 se muestra el resultado de una serie de ejecuciones con un tamaño de matriz fija, 3000×3000, lo suficientemente grande como para que los 36 procesos utilizados, tengan la suficiente capacidad de cómputo en cada iteración. Se puede observar como la mejora al realizar un reparto de la carga balanceado según las capacidades computacionales de cada nodo, es cada vez mayor. El tiempo por iteración es más corto,ya que, la carga está mejor distribuida. Por tanto, la mejoría es más notable cuantas más iteraciones. En el último punto, 10 0000 iteraciones, se alcanza la aceleración máxima de un 110 % con 520 segundos en el caso del reparto equitativo y 247 segundos en el caso de un reparto por pesos.

En la figura 8 se muestra el resultado de una serie de ejecuciones con un número de iteraciones fijo de 1000, y un tamaño n, variable, de la matriz $(n \times n)$. En la figura 8 podemos fácilmente apreciar, que, la mejora del un Layout por pesos, se aprecia en matrices generalmente grandes, que es, dónde un número grande de procesos, como 36, tienen una carga por

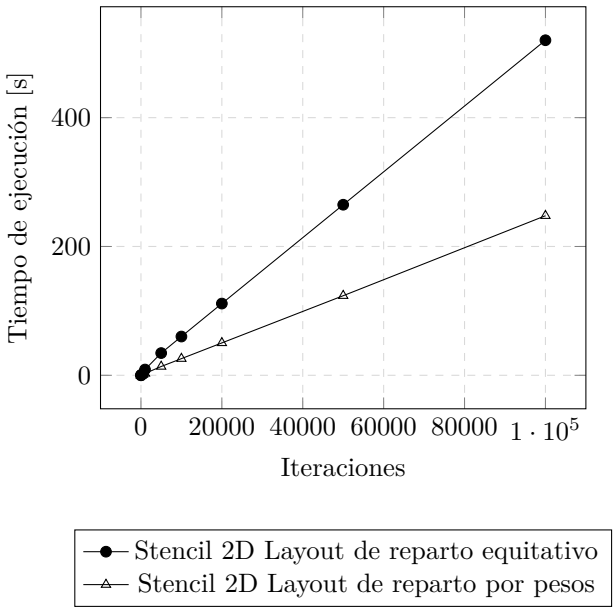


Fig. 7
Tiempo de ejecución por iteraciones, tamaño de la matriz fija: 3000×3000

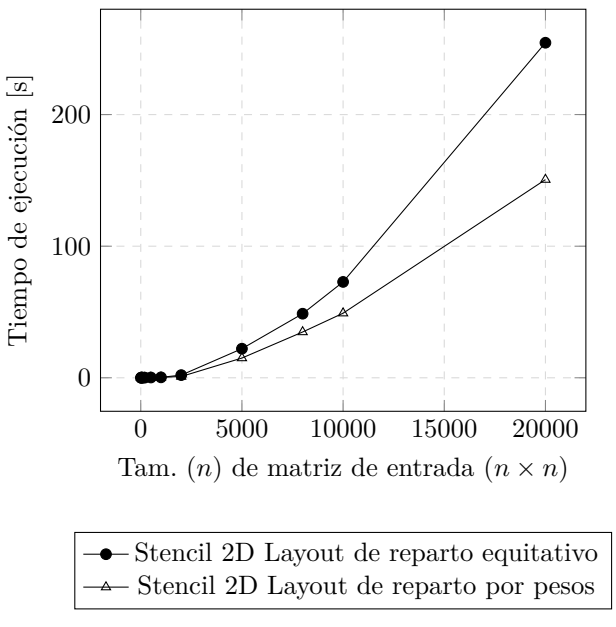


Fig. 8
Tiempo de ejecución por tamaño de la matriz, iteraciones fijas: 1000

iteración elevada. Sin embargo, en matrices pequeñas prácticamente no existe mejora. En estos casos el resultado podría empeorar. Esto es debido a que cuando la carga por iteración es pequeña, con un reparto no equitativo, podríamos dejar algunos nodos desactivados o con muy poca carga computacional comparada con otros, resultando en una espera de los nodos menos potentes a los nodos más potentes. La mejora máxima, se alcanza con la matriz de 20000×20000, una aceleración del 70 % con 154 segundos en el caso del reparto equitativo y 250 segundos en el caso de un reparto por pesos.

En general, podemos observar que, en problemas grandes el rendimiento puede llegar a ser incluso el doble, como se ha visto en la figura 7, al realizar un buen reparto de carga en un sistema heterogéneo.

VII. Conclusiones y Trabajo Futuro

Este trabajo presenta una propuesta del equilibrado de carga en sistemas heterogéneos. Se ha desarrollado un plug-in para HitMap que permite realizar una distribución de carga balanceada según los pesos que se indiquen para cada proceso. Esto permite efectuar un particionado no equitativo en sistemas en los que los nodos no poseen la misma capacidad de procesamiento. El uso del modelo que se ha propuesto, ha llegado a obtener una aceleración del 110 % con respecto a un reparto equitativo por bloques de la carga computacional.

El trabajo futuro incluye, en primer lugar, agregar un reparto entre nodos por identificación del nodo, permitiendo un grano más grueso de particionado. En segundo lugar, se pretende realizar de forma automatizada el ajuste de los pesos que cada proceso debe llevar, en tiempo de ejecución, partiendo de un cálculo previo. Por último, se pretende integrar todo este sistema de particionado automático con la librería Controller para su uso en aceleradores hardware.

Referencias

[1] Isaac Gelado, John E. Stone, Javier Cabezas, Sanjay Patel, Nacho Navarro, and Wen-mei W. Hwu, "An asymmetric distributed shared memory model for heterogeneous parallel systems," *SIGPLAN Not.*, vol. 45, no. 3, pp. 347–358, Mar. 2010.

[2] C. Luk, S. Hong, and H. Kim, "Qilin: Exploiting parallelism on heterogeneous multiprocessors with adaptive mapping," in *2009 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Dec 2009, pp. 45–55.

[3] Arturo Gonzalez-Escribano, Yuri Torres, Javier Fresno, and Diego R. Llanos, "An Extensible System for Multilevel Automatic Data Partition and Mapping," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 5, pp. 1145–1154, May 2014.

[4] "Grupo Trasgo – Trasgo Research Group (Universidad de Valladolid) - Home Page," .

[5] Didem Unat, Anshu Dubey, Torsten Hoefler, John Shalf, Mark Abraham, Mauro Bianco, Bradford L. Chamberlain, Romain Cledat, H. Carter Edwards, Hal Finkel, Karl Fuerlinger, Frank Hannig, Emmanuel Jeannot, Amir Kamil, Jeff Keasler, Paul H J Kelly, Vitus Leung, Hatem Ltaief, Naoya Maruyama, Chris J. Newburn, and Miquel Pericas, "Trends in data locality abstractions for hpc systems," *IEEE TPDS*, vol. 28, no. 10, pp. 3007–3020, Oct 2017.

[6] D.B. Loveman, "High performance fortran," *Parallel & Distributed Technology: Systems & Applications*, vol. 1, no. 1, pp. 25–42, Feb 1993.

[7] I. Foster, "Task parallelism and high-performance languages," *IEEE Parallel & Distributed Technology*, pp. 27–36, Fall 1994.

[8] B.L. Chamberlain, S.J. Deitz, D. Iten, and S-E. Choi, "User-defined distributions and layouts in Chapel: Philosophy and framework," in *2nd USENIX Workshop on Hot Topics in Parallelism*, June 2010.

[9] Antal Buss, Harshvardhan, Ioannis Papadopoulos, Olga Pearce, Timmie Smith, Gabriel Tanase, Nathan Thomas, Xiabing Xu, Mauro Bianco, Nancy M. Amato, and Lawrence Rauchwerger, "STAPL: standard template adaptive parallel library," in *SYSTOR'10*. 2010, ACM.

[10] Karl Furlinger, Colin Glass, Jose Gracia, Andreas Knupfer, Jie Tao, Denis Hunich, Kamran Idrees, Matthias Mai-

terth, Yousri Mhedheb, and Huan Zhou, "DASH: data structures and algorithms with support for hierarchical locality," in *Euro-Par 2014*. 2014, pp. 542–552, Springer.

[11] Christopher A. Bohn and Gary B. Lamont, "Load balancing for heterogeneous clusters of PCs," *Future Generation Computer Systems*, vol. 18, no. 3, pp. 389–400, Jan. 2002.

[12] J. Dongarra, T. Sterling, H. Simon, and E. Strohmaier, "High-performance computing: clusters, constellations, MPPs, and future directions," *Computing in Science Engineering*, vol. 7, no. 2, pp. 51–59, Mar. 2005.

[13] Andre R. Brodtkorb, Christopher Dyken, Trond R. Hagen, Jon M. Hjelmervik, and Olaf O. Storaasli, "State-of-the-art in Heterogeneous Computing," 2010.

[14] "Hitmap – Grupo Trasgo," .

[15] Ana Moreton–Fernandez, Hector Ortega–Arranz, and Arturo Gonzalez–Escribano, "Controllers: An abstraction to ease the use of hardware accelerators," *The International Journal of High Performance Computing Applications*, p. 109434201770296, May 2017.

[16] Sriram Krishnamoorthy, Muthu Baskaran, Uday Bondhugula, J. Ramanujam, Atanas Rountev, and P Sadayappan, "Effective Automatic Parallelization of Stencil Computations," in *Proceedings of the 28th ACM SIGPLAN Conference on Programming Language Design and Implementation*, New York, NY, USA, 2007, PLDI '07, pp. 235–244, ACM, event-place: San Diego, California, USA.

Análisis combinado de texture y contrast masking en HEVC

Javier Ruiz Atencia, Otoniel López Granado, Manuel Pérez Malumbres, Miguel O. Martínez-Rach[1]

Resumen— **En este artículo se han analizado dos técnicas de codificación perceptual, como son texture masking y contrast masking, aplicadas al estándar de codificación de vídeo HEVC. Para el texture masking se han adaptado algoritmos utilizados en compresión de imágenes a las características del HEVC, como es su transformada y los diferentes tamaños de bloque disponibles. Por su parte para el contrast masking se ha determinado la matriz de pesos óptima para cada tamaño de bloque. Por último se han codificado diferentes secuencias utilizando estas técnicas, obteniendo los valores BD-Rate con el objetivo de comprobar la viabilidad de su uso de forma individual y combinado.**

Palabras clave— **HEVC, perceptual, CSF, texture masking, constrast masking, DCT.**

I. Introducción

LOS codificadores de imagen y video eliminan la redundancia de información existente en las imágenes naturales utilizando técnicas de transformación al dominio de la frecuencia como la transformada discreta del coseno (DCT) del seno (DST) o wavelets (DWT), entre otras, que permiten compactar la información presente en la imagen en un conjunto reducido de coeficientes frecuenciales que serán codificados de distintas formas para reducir la cantidad de información a transmitir o almacenar. Así mismo, utilizan la cuantización como técnica para reducir el peso de dichos coeficientes, de tal manera que solo se codifican aquellos que son más importantes para la posterior reconstrucción de la imagen. El problema siempre ha sido determinar qué coeficientes proteger y no cuantizar por ser los más importantes desde el punto de vista subjetivo. La mayoría de los estándares de codificación utilizan un particionado en bloques de distintos tamaños para aplicar estas etapas de transformación y cuantización de manera repetitiva para toda la imagen.

En la codificación de imagen y video se han incorporado muchas técnicas, basadas en la percepción de la calidad de nuestro sistema visual humano (HVS). Estas técnicas tienen como objetivo mejorar la percepción subjetiva de la calidad de la imagen o el video reconstruido. La calidad de la reconstrucción se reduce como consecuencia de la etapa de cuantización, donde se produce la pérdida de información de la imagen, apareciendo artefactos o distorsiones visibles por nuestro HVS que no estaban presentes en la imagen original.

Sin embargo el HVS no es capaz de detectar, bajo ciertas circunstancias, dichas distorsiones cuando están enmascaradas por una alta textura, bajos o altos contrastes, baja o alta luminosidad, etc. Por tanto las condiciones del HVS que producen este enmascaramiento de los defectos han sido ampliamente estudiadas para ofrecer mecanismos que permitan eliminar selectivamente más información en aquellas zonas de la imagen donde los errores en la reconstrucción van a ser enmascarados.

Una de las propiedades más utilizada es la incorporación, normalmente en misma etapa de cuantización, de la Contrast Sensitivity Function (CSF) que pone de manifiesto que determinadas variaciones de contraste entre un objeto y su fondo no son percibidas por el sistema visual humano (HVS), siendo éstas dependientes de la luminancia, la distancia de visualización, entre otros factores. Otro factor de enmascaramiento lo produce la luminancia y se conoce como Luminnace Masking ya que en zonas con mucha o muy poca luminancia estos errores pasan más desapercibidos. También la presencia de textura en la imagen proporciona enmascaramiento de los errores en la reconstrucción, así un error en una superficie homogénea es más detectable que en una zona con mucha textura.

En [1] Tong propuso un modelo perceptual del enmascaramiento por textura y luminancia proponiendo a su vez un método para la clasificación de los bloques de coeficientes DCT en función del tipo de contenido, textura, borde o plano. Para realizar esta clasificación utiliza un algoritmo que se basa en el peso de los coeficientes agrupados en función de su frecuencia o posición dentro del bloque. Con el modelo de enmascaramiento y la clasificación de los bloques se determina el grado de cuantización adicional aplicable a un bloque para que las distorsiones producidas por la cuantización sigan enmascaradas.

El modelo de [1] ha sido refinado y variado por otros autores, por ejemplo en [2] Zhang, modifica el modelo de luminance maskig y el clasificador utilizando una ponderación del peso de los coeficientes del bloque en vez de usar la suma absoluta. Esto modifica el algoritmo de clasificación añadiendo también el efecto de intra-band masking. Este efecto se refiere a la tolerancia de error imperceptible dentro de la propia subbanda. Es decir, para cada coeficiente dentro del bloque de 8×8 se aplica un valor diferente de cuantización en función de la clasificación del bloque y de la posición del coeficiente.

Como se indica en [3] la mayoría de los modelos se basan en particionados de la imagen en bloques de 8×8 [2][4][5]. En [3] los autores extienden la clasificación a bloques de 16×16 para adaptarse a las

[1]Dpto. de Ingeniería de Computadores, Universidad Miguel Hernández de Elche, e-mail: mmrach@umh.es.

resoluciones de la imagen soportadas por los últimos estándares, sin embargo utilizan un modelo de clasificación en el dominio del píxel basado en el algoritmo de detección de bordes Canny, aplicando una cuantización adaptativa en función del tamaño del bloque. También son varios los autores [6][7] que utilizan una reducción a 4×4 del clasificador basado en [1].

Otro aspecto a considerar a la hora de incluir texture masking en un codificador, es la señalización en el bitstream del tipo de bloque o bien del valor de cuantización adicional a aplicar en cada bloque. La mayoría de los autores citados utiliza los umbrales definidos por el modelo JND para descartar completamente los coeficientes por debajo del umbral, de forma compatible con el estándar correspondiente, sin necesidad por tanto de enviar información adicional al decodificador.

Todos estos trabajos utilizan el PSNR como métrica de calidad al evaluar el rendimiento R/D. Pero como es sabido el PSNR no refleja fielmente la valoración perceptual de la calidad. Por eso en algunos estudios como en [3] se han realizado test subjetivos para validar su modelo JND utilizando el Difference Mean Opinion Score (DMOS) como indicador de la calidad perceptual.

En este trabajo nos hemos centrado en texture y contrast masking y su adaptación para el codificador HEVC. Utilizando como base el algoritmo de [2], se ha adaptado para la inclusión del texture masking en el codificador de video HEVC, modificando los umbrales para todos los tamaños de bloque. Utilizamos la versión que añade información adicional (multiplicador por cada bloque) al bitstream. Para reducir el volumen que ésta ocupa se ha aplicado un algoritmo simple de agrupación de valores de masking utilizando valores estadísticos y aplicándolo adaptativamente al tamaño de bloque. Se han utilizado las matrices estándar incluidas en el HEVC (perceptual weighting matrices) para incluir el contrast masking en bloques desde 8×8 a 32×32. El estándar no aplica la CSF para bloques de 4×4 por lo que nosotros proponemos la inclusión de una matriz perceptual específica para ese tamaño [8] con el fin de aumentar la tasa de compresión para la misma calidad perceptual.

El presente estudio utiliza la codificación en modo intra para imágenes y all intra para video y analiza el comportamiento combinado y por separado de la CSF y el texture masking para un particionado en bloques de tamaño fijo dese 4×4 hasta 32×32. Los resultados del rendimiento R/D se presentan utilizando el PSNR como métrica de distorsión pero también un conjunto amplio de métricas de calidad objetiva, SSIM, MSSSIM, VIFP, VIF, PSNRHVS y PSNRHVSM [9][10]. Para proporcionar los resultados de ahorro en rate se utiliza el método Bjøntegaard [11] adaptado a las distintas métricas.

El resto del artículos se estructura de la siguiente manera. En el capítulo II se desarrolla ampliamente la adaptación al estándar HEVC del texture masking. En el capítulo III se define nuestra propuesta de aplicar CSF a cada tamaño de bloque disponible. En el capítulo IV se muestran resultados de aplicar o no las técnicas de enmascaramiento para una serie de imagenes y secuencias de vídeo. Por último, en el capítulo V se resumen las conclusiones del estudio así como las líneas futuras de investigación para seguir ampliando este trabajo.

II. Adaptación de HEVC para texture masking

Como se ha mencionado en la introducción para este trabajo hemos utilizado el algoritmo clasificador de Zhang [2], cuyo diagrama de flujo se muestra en la Figura 1. Zhang asigna los siguientes valores a los umbrales del clasificador: $\mu_1 = 125$, $\mu_2 = 290$, $\mu_3 = 900$, $\alpha_1 = 7$, $\beta_1 = 5$, $\sigma = 16$ y $\kappa = 0{,}1$. Las expresiones E_1 y E_2 indican dos medidas utilizadas para determinar la presencia de bordes en un bloque mientras que $TexE$ indica su nivel de textura. Estas expresiones se obtienen mediante las Ecuaciones (1), (2) y (3).

$$E_1 = (\bar{L} + \bar{M})/\bar{H} \tag{1}$$

$$E_2 = \bar{L}/\bar{M} \tag{2}$$

$$TexE = M + H \tag{3}$$

donde M y H corresponden a la suma absoluta de los coeficientes de media frecuencia (MF) y alta frecuencia (HF) respectivamente; y $\bar{L}$, $\bar{M}$ y $\bar{H}$ corresponden al valor promedio de los coeficientes absolutos de baja (LF), media (MF) y alta frecuencia (HF) respectivamente. Los coeficientes se clasifican en función de la posición que ocupen en la matriz de coeficientes transformados de tamaño 8×8 (Figura 2).

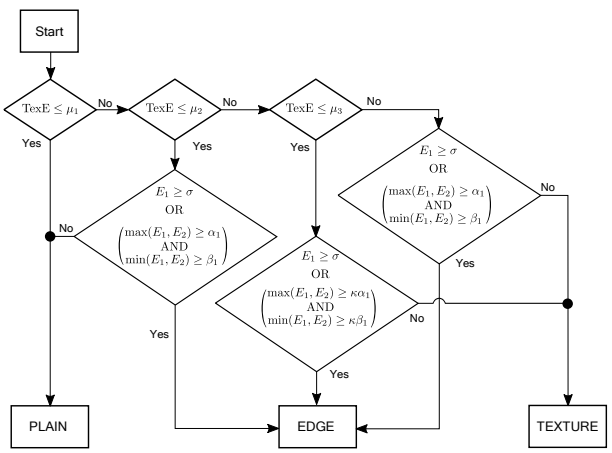


Fig. 1: Diagrama del clasificador de bloques utilizado.

Por otro lado se obtiene el factor multiplicador de masking en función de la clasificación del bloque. Para bloques TEXTURE se utiliza la Ecuación (4):

$$m(k) = (MaxElevation - 1)\frac{TexE(k) - MinE}{MaxE - MinE} + 1 \tag{4}$$

donde [1] y [2] establecen los parámetros $MaxElevation = 2{,}25$, que corresponde al valor

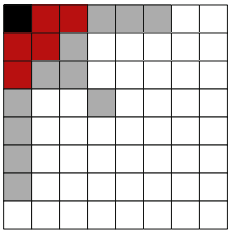


Fig. 2: Clasificación de coeficientes transformados para bloques de 8×8. Diferenciamos LF (rojo), MF (gris) y HF (blanco) como baja, media y alta frecuencia respectivamente.

máximo que puede alcanzar el factor multiplicador de masking, y las energías mínimas $MinE = 290$ y máximas $MaxE = 1800$, obtenidos empíricamente.

Tong [1] y Zhang [2], así como buena parte de los trabajos derivados de éstos basan la clasificación en bloques de tamaño 8×8 píxeles transformados mediante la Transformada Discreta de Coseno, debido a que codifican la imagen utilizando estándares cuya transformada es la DCT-II (JPEG) o alguna aproximación finita de la misma (H.264/AVC). A diferencia de éstos estándares, HEVC utiliza tamaños de bloque desde 4×4 hasta 32×32 y una aproximación finita a enteros y escalada de la DCT-II para que los vectores base de cada tamaño de matriz de transformación tengan la misma energía. Además, Tong y Zhang trabajan con la imagen original mientras que el estándar HEVC, al igual que el resto de estándares basados en el esquema Hybrid Video Coding, trabaja con el error residual de la mejor predicción calculada. Es por todo esto que en el presente trabajo se ha estudiado la adaptación del algoritmo clasificador de bloques así como de las ecuaciones para obtener el factor multiplicador de masking con el objetivo de integrarlo en el estándar HEVC.

A. Adaptación a la transformada HEVC

Debido a los cambios introducidos en la transformada del estándar HEVC resulta necesario realizar una adaptación de los umbrales con el objeto de que el algoritmo clasifique los bloques de igual forma a como lo haría utilizando la DCT-II. Para ello se ha comparado el ratio de los coeficientes de ambas transformadas aplicadas a bloques de tamaño 8×8 para un conjunto de imágenes de muestra con diferentes niveles de textura. Filtrando los valores atípicos debido a ratios entre coeficientes con energía muy baja y aplicando la media obtenemos un factor multiplicador aproximado de 16. En la Figura 3 se expone a modo de ejemplo la relación entre los coeficientes de ambas transformadas para un bloque cualquiera.

Obtenido este multiplicador se ha determinado que los umbrales a modificar son solamente μ_1, μ_2, μ_3, $MinE$, $MaxE$ y δ y cuyos valores adaptados son 2000, 4640, 14400, 4640, 28800 y 6400 respectivamente. Esto es porque las expresiones que evalúa dichos umbrales, como por ejemplo $TexE$, se obtienen a partir de la suma de coeficientes. Sin embargo, el

(a) Coeficientes Transformada HEVC

$$\begin{pmatrix} 15846 & -144 & 5 & 291 & -194 & 24 & 82 & -137 \\ -91 & -105 & 20 & 33 & -47 & 23 & 5 & 4 \\ -109 & 180 & 80 & -193 & 78 & -40 & -35 & 11 \\ -246 & 102 & 132 & -193 & 143 & -68 & -116 & 64 \\ 6 & -62 & -35 & 113 & -82 & 8 & 3 & -45 \\ -66 & -137 & 141 & 19 & 48 & 29 & -2 & 0 \\ 372 & -26 & -3 & 38 & -3 & 8 & -60 & -19 \\ 70 & 76 & -63 & -42 & -43 & 33 & 16 & 20 \end{pmatrix}$$

(b) Coeficientes Transformada DCT-II

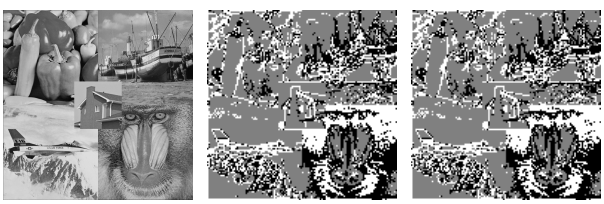
$$\begin{pmatrix} 990,4 & -9,1 & 0,4 & 18,1 & -12,1 & 1,5 & 5,1 & -8,6 \\ -5,6 & -6,5 & 1,2 & 2 & -3 & 1,5 & 0,3 & 0,2 \\ -6,4 & 11,2 & 5 & -12 & 4,9 & -2,5 & -2,3 & 0,7 \\ -15,4 & 6,4 & 8,1 & -12 & 8,9 & -4,3 & -7,4 & 4 \\ 0,4 & -3,9 & -2,2 & 7 & -5,1 & 0,5 & 0,2 & -2,9 \\ -4,2 & -8,6 & 8,9 & 1,2 & 3 & 1,8 & -0,3 & 0 \\ 23,4 & -1,8 & -0,3 & 2,6 & -0,3 & 0,5 & -3,7 & -1,2 \\ 4,4 & 4,7 & -3,9 & -2,6 & -2,7 & 2,1 & 1,1 & 1,3 \end{pmatrix}$$

(c) Coeficientes HEVC./DCT

$$\begin{pmatrix} 16 & 15,8 & 13,7 & 16,1 & 16 & 15,9 & 16 & 15,9 \\ 16,2 & 16 & 16,2 & 16,1 & 15,9 & 15,7 & 16 & 16,7 \\ 17 & 16 & 16,1 & 16,1 & 15,9 & 15,8 & 15 & 15,5 \\ 16 & 15,9 & 16,2 & 16,1 & 16,1 & 15,9 & 15,7 & 15,8 \\ 16 & 16 & 15,9 & 16,1 & 16 & 15 & 12,7 & 15,8 \\ 15,7 & 16 & 15,9 & 16,3 & 15,9 & 16,3 & 6,7 & 0 \\ 15,9 & 14,4 & 9,2 & 14,6 & 11,2 & 15,1 & 16,2 & 16,2 \\ 15,9 & 16,2 & 16,1 & 16,4 & 15,9 & 15,7 & 15 & 15,7 \end{pmatrix}$$

Fig. 3: Coeficientes transformados para un bloque de tamaño 8×8: transformada del estándar HEVC (a), transformada DCT-II (b) y ratio entre ambas transformadas (c)

resto de umbrales del algoritmo clasificador son evaluados por las expresiones E_1 y E_2, Ecuaciones (1) y (2), donde el multiplicador no es tenido en cuenta y por tanto conservan el mismo valor dado por [2]. En la Figura 4 se muestra un ejemplo del clasificador de bloques adaptado al estándar HEVC para una imagen de test. Como se puede observar, la clasificación utilizando la transformada HEVC con los umbrales adaptados, si bien no es perfecta, es bastante fiel a la realizada mediante la DCT-II.



(a) Original (b) DCT-II 8×8 (c) HEVC 8×8

Fig. 4: Clasificación de bloques 8×8 de una imagen de test (a) utilizando la transformada DCT-II (b) y la transformada HEVC con umbrales adaptados (c). En gris los bloques PLAIN, en blanco los EDGE y en negro los TEXTURE.

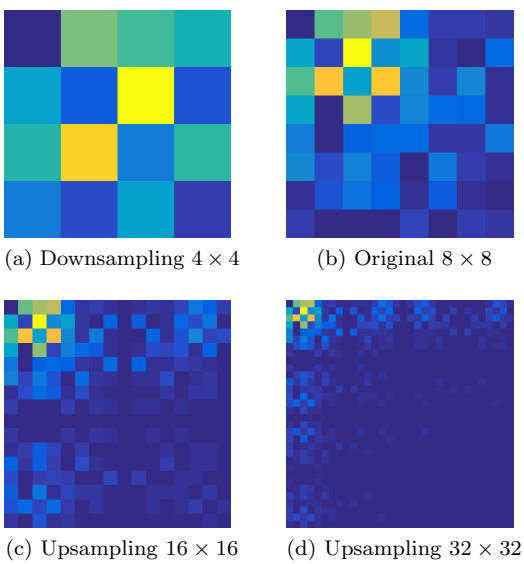
B. Adaptación a tamaños de bloque

Para adaptar el texture masking a los diferentes tamaños de bloque integrados en el estándar HEVC, se ha elaborado un análisis de los coeficientes transformados con el fin de extender el modelo propuesto por Tong [1], basado en la agrupación de estos coeficientes según su nivel frecuencial, y diferenciándonos de [3], donde los autores realizan la clasificación en el dominio del píxel mediante el algoritmo de detección

de bordes Canny.

La primera parte del análisis ha consistido en construir el modelo de agrupación en niveles frecuenciales de los coeficientes transformados. Para ello se han generado transformadas de bloques aleatorios de diferentes imágenes de muestra que se han redimensionado con el objetivo de que los bloques a transformar sean visualmente idénticos para cada tamaño de bloque. Las imágenes originales se han utilizado para la transformada 8×8; para la transformada de tamaño 4×4 se ha reducido a la mitad la imagen original mientras que para las transformadas 16×16 y 32×32 se ha aumentado en dos y cuatro veces el tamaño de la imagen respectivamente.

A partir de este análisis se ha observado cierta correlación entre el peso que tienen los coeficientes de la transformada 8×8 y el peso de esos mismos coeficientes para las transformadas de tamaño superiores. Esto era de esperar debido a las propiedades de compactación de energía de la DCT. Sin embargo para tamaños de 4×4 resulta más complejo determinar qué coeficientes determinan los diferentes niveles frecuenciales. Es por eso que en este trabajo no se ha aplicado texture masking en bloques de este tamaño.



(a) Downsampling 4×4 (b) Original 8×8

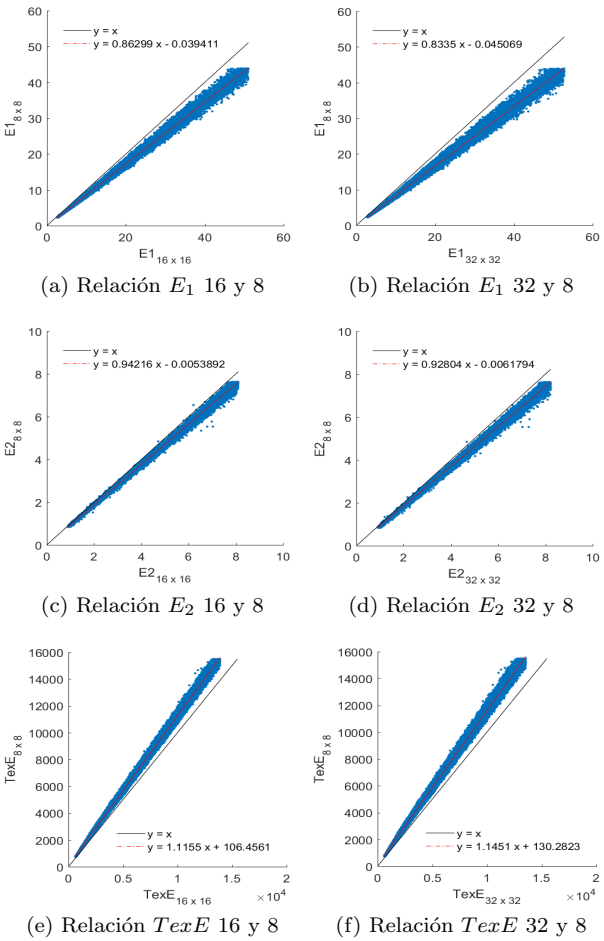
(c) Upsampling 16×16 (d) Upsampling 32×32

Fig. 5: Mapa de calor de los coeficientes AC absolutos normalizados de un mismo bloque reescalado usando la transformada DCT-II.

En la Figura 5 se muestra a modo de ejemplo los coeficientes transformados, representados como un mapa de calor, para cada tamaño disponible de un mismo bloque utilizando las redimensiones oportunas. Para una mejor visualización se ha anulado la componente DC. Comprobamos como el patrón que dan los pesos en la imagen original (b) se repite para los tamaños superiores (c) y (d), mientras que dicho patrón se ve alterado para bloques de tamaño 4×4 (a).

A partir de este análisis hemos definido la clasificación utilizando los 8×8 primeros coeficientes de cada bloque para los tamaños 16×16 y 32×32.

Una vez se tiene la clasificación de los coeficientes se ha realizado otro estudio para determinar la relación entre estos mismos coeficientes para diferentes tamaños de bloque. Para ello se ha seguido la metodología vista en el apartado II-A, esto es, se ha utilizado un conjunto variado de imágenes de muestra para realizar el particionado 8×8, y a su vez se han redimensionado para que visualmente el particionado en bloques de 16×16 y 32×32 sea equivalente al particionado de la imagen original. Hecho esto se ha extraído de cada bloque los valores E_1, E_2 y $TexE$ y se han representado para ver su relación.



(a) Relación E_1 16 y 8 (b) Relación E_1 32 y 8

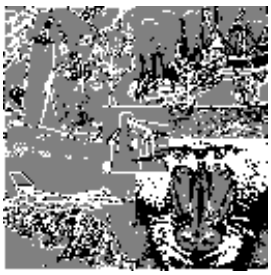
(c) Relación E_2 16 y 8 (d) Relación E_2 32 y 8

(e) Relación $TexE$ 16 y 8 (f) Relación $TexE$ 32 y 8

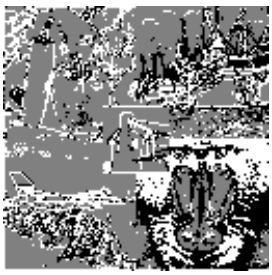
Fig. 6: Estudio estadístico de la relación entre el valor de las expresiones E_1 (a,b), E_2 (c,d) y $TexE$ (e,f) para bloques de tamaño 16×16 y 32×32 frente a bloques de tamaño 8×8

En la Figura 6 se muestra el resultado de dicho análisis. Como se puede observar existe una relación lineal entre las expresiones para tamaños de bloque 16×16 y 32×32 con respecto al tamaño 8×8. A partir de estos datos se ha extraído la recta de regresión para cada una las expresiones, aplicándose al algoritmo clasificador y dando como resultado una clasificación prácticamente igual a la clasificación original, tal y como se observa en la Figura 7.

Para obtener el factor multiplicativo de masking también es necesario realizar una adaptación de sus umbrales a los tamaños de bloque superiores a 8×8. El factor multiplicativo es un valor numérico que modifica (cuantiza) los coeficientes AC de la transformada de cada bloque en función de la clasificación del

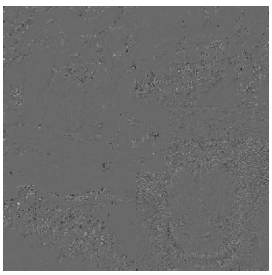


(a) HEVC 16×16

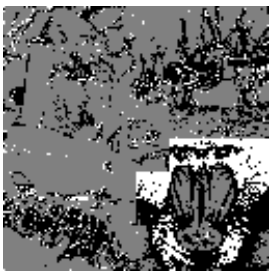


(b) HEVC 32×32

Fig. 7: Clasificación de una imagen de test utilizando la transformada HEVC con umbrales adaptados para tamaño 16×16 (a) y 32×32 (b).



(a) Imagen residual



(b) Clasificación imagen residual

Fig. 8: Imagen residual de test para el estudio del *masking* para un tamaño de bloque de 8×8 correspondiente a la Figura 4 (a)

mismo y de su nivel de energía de textura. Para este trabajo se ha decidido utilizar el esquema propuesto por Tong, explicado al comienzo del Apartado II.

Como ya hemos visto, los umbrales $MinE$, $MaxE$ y $\delta_{threshold}$ se han adaptado para la transformada HEVC y bloques de tamaño 8×8, sin embargo estos valores deben ajustarse también para bloques de tamaño superior.

En la Tabla I se muestran los umbrales adaptados para obtener el valor de masking.

TABLA I: Valor de los umbrales para la obtención del factor de masking a diferentes tamaños de bloque

Variables umbral	**Tamaño 8x8**	**Tamaño 16x16**	**Tamaño 32x32**
$MinE$	4640	4054	3925
$MaxE$	28800	25160	24364
$\delta_{threshold}$	6400	6189	6136

C. Aplicación al residuo

Una vez se tiene el algoritmo clasificador adaptado a la transformada HEVC así como a diferentes tamaños de bloque se ha estudiado su implementación para las imágenes residuales, ya que en los apartados anteriores se ha aplicado la transformada a imágenes originales.

Primero se ha intentado buscar una relación entre las expresiones E_1, E_2 y $TexE$ para los coeficientes transformados de imágenes originales e imágenes residuales de forma análoga al estudio realizado en el Apartado II-B. El resultado de este estudio indica que no es posible la adaptación de los coeficientes ya que hay una dispersión elevada.

Eso nos hizo plantearnos si realmente era necesario obtener la misma clasificación para los bloques residuales que si se estuvieran evaluando los bloques originales. La imagen residual tiene otros niveles de textura y un rango dinámico que varía entre -255 y 255 (para imágenes con 8 bits de profundidad), pero para el clasificador de bloques sus coeficientes transformados son como los de cualquier otra imagen natural. En la Figura 8 (a) se muestra error residual de la predicción correspondiente a la imagen de test utilizada en este artículo, mientras que en (b) se tiene el resultado de la clasificación de sus bloques transformados.

Si observamos con detenimiento (a) podemos apreciar cierta similitud con su imagen original: cuanto mayor nivel de textura tiene el bloque original mayor es el error de predicción, lo cual da la apariencia de textura a la imagen residual, e inversamente ocurre lo mismo para los bloques planos, cuyo residuo carece de textura debido a la facilidad de predicción usando los modos Planar y DC del estándar HEVC. Es por esto por lo que la clasificación de los bloques TEXTURE y PLAIN en el residuo es muy similar a la obtenida para la imagen original.

En cambio no ocurre lo mismo con los bloques clasificados como EDGE, ya que comprobamos como la mayoría de estos bloques han dejado de detectarse, pasando a ser en su mayoría bloques TEXTURE y en algunos casos PLAIN. Esto ocurre porque el estándar HEVC tiene 35 modos de predicción, de los cuales 33 corresponden a predicciones angulares, es decir, predicen la presencia de un borde en 33 direcciones angulares diferentes. Si se tienen bloques con bordes fuertemente marcados es de esperar que el codificador selecciones un modo de predicción angular de forma que el error residual tenga un nivel de borde muy bajo o, en caso de tener una predicción muy buena, inexistente.

Llegando a esta conclusión hemos optado por no adaptar o reajustar los umbrales o expresiones del algoritmo clasificador de bloques para obtener el mismo resultado que la clasificación en imágenes originales, ya que estamos clasificando el residuo y no la imagen original.

III. Adaptación de HEVC para contrast masking

El contrast masking o enmascaramiento por contraste es una técnica de compresión perceptual que explota el efecto visual por el cual el ojo humano no es capaz de percibir pequeñas variaciones de luminancia cuando el fondo de la escena es lo suficientemente claro u oscuro. Numerosos estudios se han encargado de caracterizar lo que llamamos la Función de Sensibilidad al Contraste (Contrast Sensitivity Function, CSF) a partir de mediciones del

umbral de contraste en humanos para diferentes frecuencias espaciales [12][13][14], sin embargo el modelo más extendido en el ámbito de la codificación de imagen y vídeo es el de Mannos y Sakrison [15].

El estándar HEVC incorpora cuantización dependiente de la frecuencia, esto es, aplica diferente nivel de cuantización dependiendo del peso que tengan los coeficientes en el dominio frecuencial basándose para ello en el modelo del Sistema Visual Humano (HVS) [16][17], utilizando para ello matrices de cuantización no-uniformes, también llamadas matrices de pesos. Para codificación intra-luminancia se define esta matriz de cuantización únicamente para tamaños de bloque 8×8, mientras que para tamaños superiores (16×16 y 32×32) las matrices se obtienen mediante sobre-muestreo y replicación. Para el caso de los bloques de tamaño 4×4 el estándar no define matriz de cuantización de pesos.

En este trabajo se ha introducido la matriz de cuantización de pesos para tamaños de bloque 4×4 con el objetivo de incrementar el nivel de compresión, mientras que para el resto de bloques se ha utilizado la matriz que incorpora el estándar. Para obtener la matriz de pesos 4×4 propuesta se ha empleado el estudio realizado por Martínez-Rach [8], y cuyo valor final es mostrado en la Figura 9 (b).

$$\begin{bmatrix} 16 & 16 & 16 & 16 \\ 16 & 16 & 16 & 16 \\ 16 & 16 & 16 & 16 \\ 16 & 16 & 16 & 16 \end{bmatrix}$$

(a) HEVC QM 4×4

$$\begin{bmatrix} 16 & 16 & 20 & 32 \\ 16 & 17 & 21 & 37 \\ 20 & 21 & 29 & 55 \\ 32 & 37 & 55 & 115 \end{bmatrix}$$

(b) Propuesta QM 4×4

$$\begin{bmatrix} 16 & 16 & 16 & 16 & 17 & 18 & 21 & 24 \\ 16 & 16 & 16 & 16 & 17 & 19 & 22 & 25 \\ 16 & 16 & 17 & 18 & 20 & 22 & 25 & 29 \\ 16 & 16 & 18 & 21 & 24 & 27 & 31 & 36 \\ 17 & 17 & 20 & 24 & 30 & 35 & 41 & 47 \\ 18 & 19 & 22 & 27 & 35 & 44 & 54 & 65 \\ 21 & 22 & 25 & 31 & 41 & 54 & 70 & 88 \\ 24 & 25 & 29 & 36 & 47 & 65 & 88 & 115 \end{bmatrix}$$

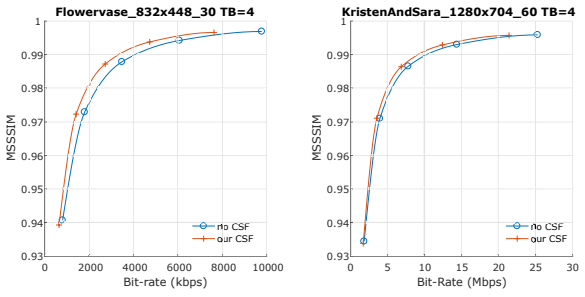
(c) HEVC QM 8×8

Fig. 9: Matrices de cuantización por defecto para tamaños de bloque 4×4 (a) y 8×8 (c), así como nuestra propuesta no-uniforme para tamaño de bloque 4×4 (b).

A modo de ejemplo se muestra en la Figura 10 una comparativa entre aplicar o no nuestra propuesta de matriz de pesos a bloques de tamaño 4×4. Se ha demostrado que la métrica PSNR es poco consistente en lo referente a la percepción del HVS [18]. En la figura comparamos ambas curvas utilizando la métrica de calidad perceptual MS-SSIM [19], donde observamos que aplicando CSF se obtiene un mejor resultado.

IV. Resultados

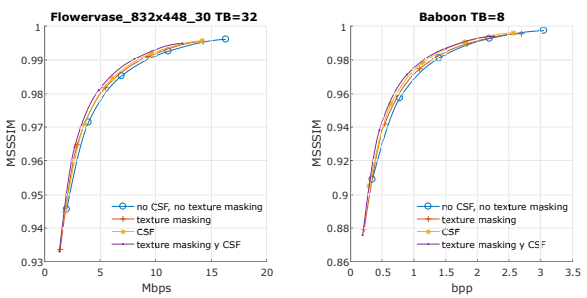
Para obtener los resultados de evaluación de las técnicas de enmascaramiento analizadas en este trabajo se ha utilizado un emulador Intra-HEVC desarrollado en Matlab por la Universidad Miguel Hernández de Elche y cuyos resultados se han validado con el software de referencia HM 14.0 [20]. Se han escogido una serie de imágenes y secuencias de vídeo con tamaños y nivel de textura variada. Para cada secuencia de vídeo se ha codificado 1 segundo de metraje, cuyo número de fotogramas varía en función de los FPS.



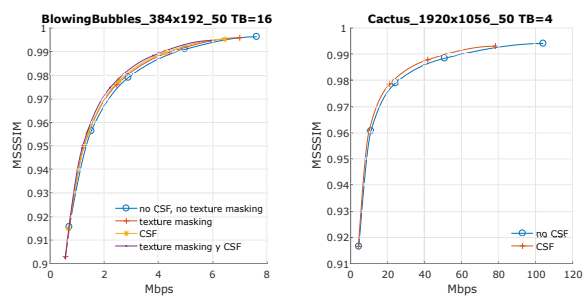
(a) (b)

Fig. 10: Comparativa de curvas R-D entre aplicar o no nuestra propuesta de CSF para varias secuencias de vídeo particionadas en bloques fijos de tamaño 4×4.

Las curvas Rate-Distortion se han obtenido aplicando texture y contrast masking de forma individual y conjuntamente. Para el caso de los bloques de tamaño 4×4 únicamente se ha aplicado contrast masking puesto que no se ha desarrollado el texture masking tal y como se ha explicado en el Apartado II-B. Las curvas donde se ha aplicado texture masking (individual o junto con contrast masking) incluyen el overhead de bits producto de enviar la matriz de masking por el bitstream. En la Figura 11 se muestran ejemplos de curvas rate-distortion para cada tamaño de bloque disponible.



(a) Curva MSSSIM - Mbps para tamaño de bloque 32×32 (b) Curva MSSSIM - bpp para tamaño de bloque 16×16



(c) Curva MSSSIM - Mbps para tamaño de bloque 8×8 (d) Curva MSSSIM - Mbps para tamaño de bloque 4×4

Fig. 11: Ejemplo de curvas Rate-Distortion para diferentes imágenes y secuencias de vídeo.

Con el fin de extraer un valor numérico para determinar la mejora o no de ahorro en rate se utiliza el cálculo Bjøntegaard-Delta Rate (BD-Rate) [11] sobre la métrica de calidad MS-SSIM para cada una de

las secuencias analizadas, donde los valores negativos indican un ahorro en rate. En las Tablas II a V se muestra el resultado del estudio realizado para cada uno de los tamaños de bloque definidos.

TABLA II: Resultados BD-Rate (%) para métrica perceptual MS-SSIM y tamaño de bloque 32×32.

Secuencia test	Texture masking	Contrast masking	Texture y contrast masking
Baboon (imagen)	-2.06	-8.90	-10.39
Lena (imagen)	-5.25	-3.81	-9.49
BlowingBubbles	-2.03	-5.01	-6.57
Flowervase	-7.56	-7.56	-13.92
RaceHorses	-3.08	-4.34	-7.25
KristenAndSara	-9.09	-2.41	-10.91
Kimono	-4.18	-0.50	-4.50
Cactus	-6.90	-2.55	-8.84
Promedio	-5.02	-4.39	-8.98

TABLA III: Resultados BD-Rate (%) para métrica perceptual MS-SSIM y tamaño de bloque 16×16.

Secuencia test	Texture masking	Contrast masking	Texture y contrast masking
Baboon (imagen)	-2.15	-9.15	-10.86
Lena (imagen)	-8.60	-3.00	-10.47
BlowingBubbles	-4.80	-5.91	-9.10
Flowervase	-10.36	-8.68	-17.06
RaceHorses	-5.35	-5.31	-9.21
KristenAndSara	-9.29	-3.08	-11.46
Kimono	-2.25	-0.67	-2.63
Cactus	-7.35	-3.17	-9.54
Promedio	-6.27	-4.87	-10.04

TABLA IV: Resultados BD-Rate (%) para métrica perceptual MS-SSIM y tamaño de bloque 8×8.

Secuencia test	Texture masking	Contrast masking	Texture y contrast masking
Baboon (imagen)	-6.06	-11.03	-15.02
Lena (imagen)	-3.94	-5.59	-9.51
BlowingBubbles	5.47	-8.01	-10.84
Flowervase	-10.62	-10.70	-18.64
RaceHorses	-3.75	-7.53	-8.60
KristenAndSara	-6.51	-4.48	-10.01
Kimono	-1.03	-1.18	-1.97
Cactus	-5.99	-4.87	-9.50
Promedio	-5.42	-6.67	-10.51

Podemos observar como para todos los tamaños de bloque aplicar técnicas de masking mejora el ahorro en rate con respecto a no utilizarla, aunque existen diferencias notables entre las distintas secuencias utilizadas. Esto ocurre porque se está comprimiendo la imagen en función de la escena con técnicas perceptuales, y por tanto la eficiencia de utilizar estas técnicas dependerá del nivel de textura (texture masking) y de las componentes frecuenciales de los diferentes objetos de la escena (contrast masking). Cabe destacar también que el uso de aplicar ambas técnicas de enmascaramiento vistas en este trabajo produce siempre un mejor resultado que aplicar únicamente una de ellas.

TABLA V: Resultados BD-Rate (%) para métrica perceptual MS-SSIM y tamaño de bloque 4×4.

Secuencia test	Contrast masking
Baboon (imagen)	-17.62
Lena (imagen)	-9.12
BlowingBubbles	-14.57
Flowervase	-19.02
RaceHorses	-13.15
KristenAndSara	-9.69
Kimono	-1.97
Cactus	-9.12
Promedio	-11.78

En lo que respecta a utilizar CSF en bloques de tamaño 4×4 tal y como se observa en la Tabla V podemos concluir que su uso está justificado, y es donde se obtiene un mejor BD-Rate con respecto a los otros tamaños de bloque.

V. Conclusiones y trabajo futuro

Las técnicas de compresión basadas en el sistema visual humano (HVS) como son el texture y contrast masking han sido utilizadas durante años demostrando que son una herramienta capaz de reducir el rate sin que ello perjudique a la calidad de la imagen. En este trabajo se han adaptado dichas técnicas a las características propias del estándar HEVC y se ha demostrado que su uso sigue siendo válido para reducir la tasa de bits.

Como trabajos futuros se pretende (a) seguir adaptando las técnicas de masking para tamaños superiores a 32×32 para utilizarlos en los nuevos estándares, como son JEM y VVC, (b) proponer algoritmos para reducir el overhead producido por el texture masking, y (c) incluir otras técnicas de compresión perceptual, como por ejemplo el luminance masking.

En cuanto al estándar HEVC, como se ha indicado anteriormente, se ha utilizado una herramienta en Matlab cuyo particionado se determina en bloques de tamaño fijo. Queda pendiente aplicar las técnicas de masking para el particionado Quad-tree propio del estándar.

Por último, por mucho que la métrica MS-SSIM esté optimizada para comparaciones perceptuales de imágenes, queda pendiente realizar estudios subjetivos DMOS.

Agradecimientos

Este trabajo ha sido financiado por el Ministerio de Ciencia, Innovación y Universidades con referencia RTI2018-098156-B-C54 cofinanciado con fondos FEDER (MINECO/FEDER/UE).

Referencias

[1] H.H.Y. Tong and A.N. Venetsanopoulos, "A perceptual model for jpeg applications based on block classification, texture masking, and luminance masking," in *Image Processing, 1998. ICIP 98. Proceedings. 1998 International Conference on*, Oct 1998, pp. 428–432 vol.3.

[2] X.H. Zhang, W.S. Lin, and P. Xue, "Improved estimation for just-noticeable visual distortion," *Signal Processing*, vol. 85, no. 4, pp. 795 – 808, 2005.

[3] L. Ma and K. N. Ngan, "Adaptive block-size transform based just-noticeable difference profile for videos," in *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*, May 2010, pp. 4213–4216.

[4] Z. Wei and K. N. Ngan, "Spatio-temporal just noticeable distortion profile for grey scale image/video in dct domain," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 19, no. 3, pp. 337–346, March 2009.

[5] Yuhong Wang, Chi Zhang, and Sukesh Kaithaapuzha, "Visual masking model implementation for images & video," 2010.

[6] X. Gong and H. Lu, "Towards fast and robust watermarking scheme for h.264 video," in *2008 Tenth IEEE International Symposium on Multimedia*, Dec 2008, pp. 649–653.

[7] C. Mak and K. N. Ngan, "Enhancing compression rate by just-noticeable distortion model for h.264/avc," in *2009 IEEE International Symposium on Circuits and Systems*, May 2009, pp. 609–612.

[8] Miguel Onofre Martínez-Rach, *Perceptual image coding for wavelet based encoders*, Ph.D. thesis, Universidad Miguel Hernández, Dec. 2014.

[9] H. R. Sheikh and A. C. Bovik, "Image information and visual quality," *IEEE Transactions on Image Processing*, vol. 15, no. 2, pp. 430–444, Feb 2006.

[10] Philippe Hanhart, "VQMT: Video Quality Measurement Tool," `https://mmspg.epfl.ch/downloads/vqmt/`, 2013.

[11] G Bjontegaard, "Calculation of average psnr differences between rd-curves," *Proceedings of the ITU-T Video Coding Experts Group (VCEG) Thirteenth Meeting*, 01 2001.

[12] K. N. Ngan, K. S. Leong, and H. Singh, "Adaptive cosine transform coding of images in perceptual domain," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 37, no. 11, pp. 1743–1750, Nov 1989.

[13] B. Chitprasert and K. R. Rao, "Human visual weighted progressive image transmission," *IEEE Transactions on Communications*, vol. 38, no. 7, pp. 1040–1044, July 1990.

[14] N. Nill, "A visual model weighted cosine transform for image compression and quality assessment," *IEEE Transactions on Communications*, vol. 33, no. 6, pp. 551–557, June 1985.

[15] J. Mannos and D. Sakrison, "The effects of a visual fidelity criterion of the encoding of images," *IEEE Transactions on Information Theory*, vol. 20, no. 4, pp. 525–536, July 1974.

[16] Long-Wen Chang, Ching-Yang Wang, and Shiuh-Ming Lee, "Designing jpeg quantization tables based on human visual system," in *Proceedings 1999 International Conference on Image Processing (Cat. 99CH36348)*, Oct 1999, vol. 2, pp. 376–380 vol.2.

[17] Yoshitaka Morigami Munsi Haque, Ali Tabatabai, "HVS model based default quantization matrices," in *Meeting report of the seventh meeting of the Joint Collaborative Team on Video Coding (JCT-VC)*, Nov 2011.

[18] Zhou Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: from error visibility to structural similarity," *IEEE Transactions on Image Processing*, vol. 13, no. 4, pp. 600–612, April 2004.

[19] Z. Wang, E. P. Simoncelli, and A. C. Bovik, "Multiscale structural similarity for image quality assessment," in *The Thrity-Seventh Asilomar Conference on Signals, Systems Computers, 2003*, Nov 2003, vol. 2, pp. 1398–1402 Vol.2.

[20] Fraunhofer Institute for Telecommunications, "HM Reference Software Version 14.0," `https://hevc.hhi.fraunhofer.de/svn/svn_HEVCSoftware/tags/HM-14.0/`, 2014.

Conversión de superficies NURBS a superficies Bézier en la GPU

Lois A. Gómez[1], Sergio Vázquez[2], Margarita Amor[3]

Resumen— **Las superficies NURBS son uno de los formatos más utilizados para la representación y diseño de datos en productos industriales debido a su capacidad para representar modelos geométricos complejos. Las GPUs actuales no proporcionan un procedimiento directo para realizar el *rendering* de una superficie NURBS. Existen dos aproximaciones para realizar el *rendering* de las superficies NURBS: conversión en la CPU de las superficies NURBS a otro tipo de representación o modificación del *pipeline* de las GPUs para realizar el *rendering* directo. En este trabajo se presenta una propuesta de conversión de superficies NURBS a superficies Bézier directamente en la GPU, aprovechando la capacidad de procesamiento paralelo de las tarjetas gráficas actuales.**

Palabras clave— **Superficies NURBS, Superficies Bézier, GPU, Direct3D, Compute Shader.**

I. Introducción

LAS superficies NURBS (*Non-Uniform Rational B-Splines*) [1] son uno de los formatos más utilizados para el diseño de productos industriales y en general, en aplicaciones CAD/CAM/CAE. Esto se debe a que las superficies NURBS presentan una gran flexibilidad en el diseño. Sin embargo, las GPUs (*Graphical Processing Units*) actuales no cuentan con un procedimiento directo para realizar el *rendering* de superficies NURBS. Existen dos aproximaciones principalmente. La primera aproximación está basada en la transformación de las superficies NURBS a otro tipo de representación, cómo pueden ser las superficies Bézier [2] o las superficies de subdivisión Catmull-Clark [3]. Estas representaciones pueden ser *renderizadas* directamente en la GPU ([4], [5], [6], [7]), tienen la desventaja de que no permiten el manejo interactivo de superficies deformables, debido al alto coste computacional asociado a la transformación de una superficie NURBS en *patches* Bézier, que tienen que ser realizados repetidamente durante la deformación de la superficie.

La segunda aproximación ([8], [9], [10]) se basa en el *rendering* de las superficies NURBS directamente en la GPU. En [8] y [9] se muestran unos nuevos métodos para evaluar y mostrar superficies NURBS recortadas siguiendo el planteamiento de [11] y [12] respectivamente, generando un conjunto de *grids* en la CPU que indican los puntos de evaluación para los distintos niveles de detalle y enviándolos a la GPU, donde son almacenados como texturas. En RPNS (*Rendering Pipeline for NURBS Surface*) [10] se presenta una propuesta que realiza la evaluación directa de superficies NURBS en la GPU sin la necesidad de realizar una descomposición previa a superficies Bézier. Esta propuesta está basada en una nueva primitiva, `KSQuad`, manteniendo las principales propiedades geométricas de las superficies NURBS y consiguiendo *real-time rendering*. Sin embargo, se propone para ellos un nuevo *pipeline* que no está disponible en las GPUs actuales.

En Direct3D 11, incluido en DirectX 11, se introdujo una nueva etapa en el *pipeline* gráfico, `Compute Shader` [13], un *pipeline* de computación, también conocido como `DirectCompute`, que fue introducido para poder realizar cálculos diferentes al paradigma clásico de computación gráfica. De esta manera, permite emplear la potencia y el paralelismo de las GPUs actuales para realizar tareas de propósito general (GPGPU) directamente en Direct3D.

En este trabajo se proponen una serie de propuestas que permiten la conversión de superficies NURBS en superficies Bézier directamente en la GPU usando el algoritmo *knot refinement* [1], consiguiendo un *rendering* interactivo.

II. Conversión de superficies NURBS a superficies Bézier en GPU

Las superficies NURBS (Figura 1) son una generalización de los B-splines y las superficies Bézier, con la principal diferencia en el peso de los puntos de control, de manera que las NURBS son racionales. Las superficies NURBS se obtienen como el producto tensor de dos curvas NURBS siendo la superficie resultante denotada por su grado, un conjunto de puntos de control ponderados y dos vectores `knot`. La ecuación que define una superficie NURBS de grado (p, q) usando dos parámetros independientes u y v es:

$$S(u,v) = \frac{\sum_{i=0}^{n}\sum_{j=0}^{m} N_{i,p}(u)N_{j,q}(v)w_{i,j}B_{i,j}}{\sum_{i=0}^{n}\sum_{j=0}^{m} N_{i,p}(u)N_{j,q}(v)w_{i,j}}, \; 0 \leq u,v \leq 1 \tag{1}$$

siendo $B_{i,j}$ los puntos de control, $w_{i,j}$ los pesos, $n+1$ y $m+1$ el número de puntos de control respectivos en las direcciones paramétricas u y v, y $N_{i,p}$ y $N_{j,q}$ las funciones básicas del B-spline no relacional definido como dos vectores *knot* de $p+n+1$ y $q+m+1$

[1]University of A Coruña, 15071, A Coruña, Spain, e-mail: `lois.gomez@udc.es`.
[2]CITIC-Research Center of Information and Communication Technologies, University of A Coruña, 15071, A Coruña, Spain, e-mail: `sergio.vazquez@udc.es`.
[3]CITIC-Research Center of Information and Communication Technologies, University of A Coruña, 15071, A Coruña, Spain, e-mail: `margarita.amor@udc.es`.

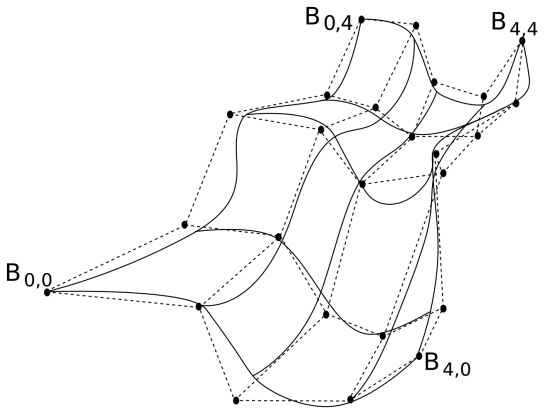


Fig. 1: Superficie NURBS.

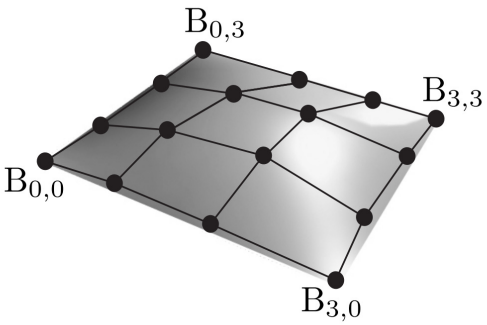


Fig. 2: Superficie Bézier.

elementos respectivamente:

$$U = \{\underbrace{0,....0}_{p+1}, x_{p+1}, ..., x_{r-p-1}\underbrace{1,...,1}_{p+1}\} \quad (2)$$

$$V = \{\underbrace{0,....0}_{q+1}, y_{q+1}, ..., x_{s-q-1}\underbrace{1,...,1}_{q+1}\} \quad (3)$$

donde $r = n + p + 1$ y $s = q + m + 1$.

Por su parte, las superficies Bézier (ver Figura 2) son un caso particular de NURBS, que son usadas normalmente debido a su estructura regular y su simplicidad. Una superficies Bézier de grado (n, m) está controlado por un conjunto de puntos de control mediante la ecuación:

$$Q(u,v) = \sum_{i=0}^{n}\sum_{j=0}^{m} B_{i,j}J_{n,i}(u)J_{m,j}(v),\ 0 \leq u, v \leq 1 \quad (4)$$

donde $J_{n,i}(u)$ y $J_{m,j}(v)$ son los polinomios de Bernstein en las direcciones paramétricas u y v, mientras que $B_{i,j}$ son los vértices de una malla de puntos de control y el número de puntos de control de u y v son $n+1$ y $m+1$ respectivamente.

Una superficie NURBS puede descomponerse en segmentos de superficies Bézier usando el algoritmo llamado *knot refinement* [1]. En el proceso de transformación de esta técnica es necesario tener en cuenta

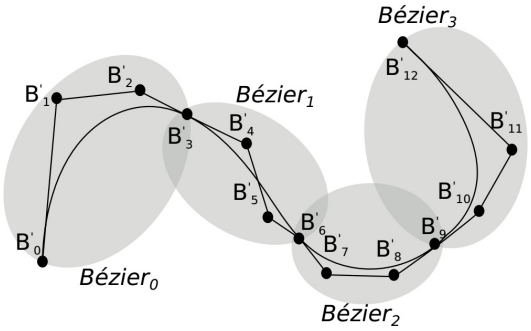


Fig. 3: Curva NURBS descompuesta en curvas Bézier.

la existencia de dos direcciones (u y v) y el vector *knot* asociado a cada dirección. Para realizar la conversión se inserta cada uno de los nodos interiores hasta que su multiplicidad es equivalente al grado y se calculan los nuevos puntos de control asociados. Hay que tener en cuenta que hay dos direcciones diferentes, debido a esto primero se realiza la transformación a lo largo de una de las direcciones y luego, aprovechando los datos obtenidos, se vuelve a aplicar el algoritmo en la otra dirección.

Partiendo de una malla de puntos $B_{i,j}(u,v)(i \in [0,n], j \in [0,m])$, denotada como $Q_{i,j,0}(u,v)$, donde el vector *knot* tiene la forma $U = \{x_0, ..., x_k, x_{k+1}, ..., x_m\}$ y el grado respectivo de las curvas es p en la dirección u y q en la dirección v. Cada uno de los *knot* internos del vector *knot* U, $x_k(k \in [0,n])$ de multiplicidad s tiene que ser insertado $(p-s)$ veces a lo largo de una curva v-paramétrica. El conjunto de puntos de control a considerar inicialmente es: $Q_{k-p-s,j,0}, ..., Q_{k-s,j,0}$. De éstos, los primeros $(s+1)$ puntos permanecen intactos $Q_{k-p-s,j,0}, ..., Q_{k-p-1,j,0}$, y el resto se sustituirá por las diagonales superior e inferior de dicho diagrama. Después de $(p-s)$ inserciones, se pueden recoger los puntos de control $Q^b_{t,j}(x_k, v)(t|in[k-p; k-s])$ para una v determinada. El nuevo vector *knot* será:

$$\{Q^{b_u}_{0,j,0}, ..., Q^{b_u}_{k-p-1,j,0},$$
$$\overbrace{, Q^{b_u}_{k-p,j,0}, ..., Q^{b_u}_{z,j,p-s}}^{diagonal\ superior}, \underbrace{Q^{b_u}_{z,j,p-s-1}, ..., Q^{b_u}_{z,j,0}, Q^{b_u}_{z+1,j,0}}_{diagonal\ inferior},$$
$$..., Q^{b_u}_{n,j,0}\}$$
$$\text{siendo } z = k - s \quad (5)$$

Una vez que todos los nodos internos del vector *knot* U han sido insertados a lo largo de todas las curvas v-paramétricas se repite el proceso para cada uno de los *knots* internos del vector V considerando cada una de las curvas u-paramétricas. El resultado final es un nuevo conjunto de puntos de control que determinan cada una de las superficies Bézier.

La primera fase de la transformación de las su-

perficies NURBS, en la que se realiza la conversión en la dirección u, da como resultado unas superficies definidas por puntos de control que denominaremos *stripes*. Estas nuevas superficies presentan las características de las superficies Bézier en la dirección u, a la vez que conservan la naturaleza de las superficies NURBS en la dirección v. De esta manera, por cada superficie NURBS procesada se generarán tantas *stripes* como número de *knot spans* no nulos aparezcan en el vector *knot* asociado a la dirección u. La dimension en la dirección v de estas *stripes* se corresponde con el grado de la superficie en la dirección u más 1 ($p+1$ puntos de control), mientras que la dimensión en la dirección v permanecerá constante ($m+1$).

La segunda fase de la transformación realiza la conversión en la dirección v a las *stripes* generadas en la fase anterior, obteniendo como resultado las superficies Bézier. Al igual que en el caso anterior, con la conversión de cada *stripe* se generan tantas superficies nuevas como *knot spans* no nulos aparecen en el vector *knot* asociado a la dirección v. De esta manera, el resultado de superficies Bézier final se corresponde al producto entre el número de *knot spans* no nulos de las direcciones u y v, con dimensiones igual al grado de las superficies NURBS en cada dirección más uno ($p+1,\ q+1$).

III. Desarrollo del algoritmo en Direct3D

El *pipeline* gráfico de una API gráfica especifica como son procesados los datos en las distintas etapas para producir la imagen final. En la Figura 4 se muestra el *pipeline* gráfico de acuerdo a Direct3D 11, donde se distinguen dos tipos de etapas: etapas fijas y etapas programables. Las etapas fijas pueden ser configuradas, pero siempre realizan las mismas operaciones, en la figura se muestran con un color más oscuro. Las etapas programables permiten ejecutar *kernels*, conocidos como *shaders*, permitiendo la implementación de distintos algoritmos dotando al *pipeline* de una gran flexibilidad. La etapa de `Input Assembler` es el punto de entrada del *pipeline* gráfico, es responsable de leer desde memoria y ensamblar los datos de entrada conectando los vértices o puntos de control. El `Vertex Shader`, la primera etapa programable, lee la información generada por el `Input Assembler` y procesa los vértices uno a uno, de manera aislada. Las siguientes tres etapas funcionan de modo conjunto para lograr un sistema de *tessellation*. El `Hull Shader` es una etapa programable responsable de llamar a una función por cada primitiva en la que se determinan los factores de *tessellation*, así como ejecuta *shaders* que tienen acceso al punto de control actual y al resto de puntos de control. La etapa fija llamada `Tessellator` recibe los factores de *tessellation* generados por el `Hull Shader` y emplea un algoritmo para realizar un patrón de *tessellation* sobre la primitiva, dividiéndola en partes más pequeñas. Su salida es una serie de coordenadas baricéntricas que son pasadas al `Domain Shader`. El `Domain Shader`, el último

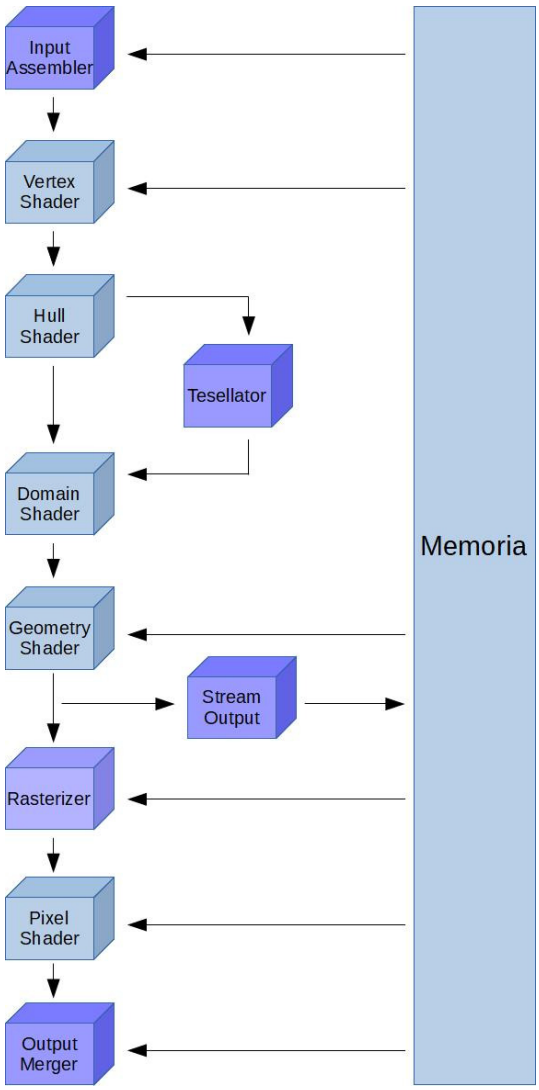


Fig. 4: Estructura del *Pipeline* gráfico según Direct3D 11.

paso en el sistema de *tessellation*, se trata de una etapa programable que recibe tanto la salida del *tessellator* junto con la de `Hull Shader` y las emplea para generar nuevos vértices. El `Geometry Shader` se trata de una etapa programable, que opera a nivel de geometría recibiendo y generando primitivas. Permite retroalimentar el *pipeline* con nuevos datos a través de la etapa de `Stream Output` mediante la generación de *buffers*. La etapa fija `Rasterizer` procesa las primitivas generadas por el `Geometry Shader`, generando fragmentos que determinan que píxeles de la imagen final abarcan cada una de las primitivas. El `Pixel Shader`, última etapa programable del *pipeline*, se invoca para cada fragmento generado por el `Rasterizer`. Esta etapa genera un color para cada una de las primitivas finales. El `Output Merger State` es una etapa fija que se encarga de "mezclar" los resultados del `Pixel Shader` y de determinar la visibilidad de los objetos para realizar el *rendering* final de la imagen.

El `Compute Shader` [13] permite realizar computación de propósito general (GPGPU) en programas basados en el *rendering* de gráficos. El `Compute`

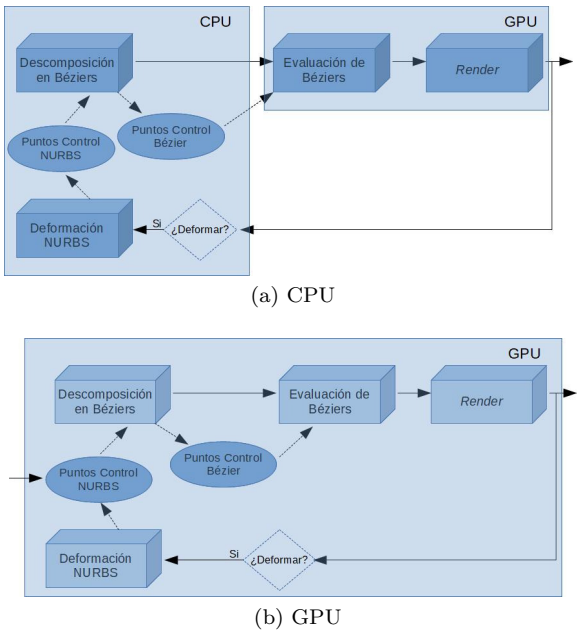
`Shader` es una etapa ajena al *pipeline* gráfico y fue incluido en Direct3D 11. Una de las nuevas características que posee es un modelo de *threading* que otorga al programador control directo sobre el comportamiento de los *threads*. Otra característica es el acceso a una memoria compartida de rápido acceso (*Group Shared Memory*), que permite la comunicación entre *threads* en tiempo de ejecución. Por último, la posibilidad de leer y escribir en los recursos de manera desordenada, que representó un cambio significativo respecto al *pipeline* gráfico, que sólo permitía el acceso a los recursos para lectura o escritura, y que provee otro medio de comunicación entre *threads*.

Al igual que el resto de APIs enfocadas a GPGPU, DirectCompute se basa en el concepto de múltiples *threads*. El `Compute Shader` es un *kernel* que es invocado por un conjunto de *threads* que están agrupados en grupos de *threads*. Los grupos son distribuidos por el *hardware* entre las unidades computacionales disponibles y, dependiendo de la cantidad de recursos requeridos, cada unidad computacional puede ser capaz de ejecutar simultáneamente varios grupos. Cada grupo tiene asignada una cantidad de memoria compartida (GSM), que permite el intercambio de datos entre los *threads* del mismo grupo.

En la Figura 5 se muestran las diferentes aproximaciones para realizar la conversión de superficies NURBS en superficies Bézier. La Figura 5a muestra la estructura de conversión de superficies tradicional, en donde la transformación se hace en la CPU, de manera que es necesario pasar por esta para poder aplicar cualquier cambio a la superficie. La Figura 5b muestra el diagrama de nuestra propuesta, en donde todo el cálculo se realiza directamente en la GPU, evitando así tener que pasar por CPU para aplicar cualquier cambio.

En este trabajo se han propuesto 3 propuestas para realizar la conversión de superficies NURBS a Bézier en GPU, `Naive`, `TextureData` y `MultipleSurfaces`. La primera propuesta, `Naive`, se basa en una implementación básica del algoritmo, en la que simplemente se llevan a cabo las operaciones necesarias. El algoritmo aplica la conversión en las dos direcciones paramétricas de las superficies, donde la primera fase se aplica en la dirección v y da como resultado los *stripes*, mientras que la segunda fase aplica la conversión a los *stripes* obtenidos en la dirección v. En la Figura 6 se muestra el pseudocódigo correspondiente a la aplicación del algoritmo. En la línea 1 se especifica que cada grupo de *threds* estará formado por un único *thread*, de manera que cada superficie se evalúa por un único *thread* y cada grupo evalúa una superficie diferente. De la línea 4 a la 6 se realiza la lectura de los datos. En la línea 8 se realiza la primera fase, la conversión en la dirección u. En las líneas 9-10 se realiza la conversión en la dirección v, la segunda fase.

La segunda propuesta, `TextureData`, busca solucionar una limitación de memoria de la propuesta



(a) CPU

(b) GPU

Fig. 5: Estructura de la conversión superficies NURBS a Bézier.

```
1  [numthreads(1,1,1)]
2  void main(uint3 id :
   ↪  SV_DispatchThreadID)
3  {
4    struct data : datos de la superficie
5    uint numControlPoints : número puntos
     ↪  control de la superficie
6    uint numStripes : número de stripes
     ↪  generados
7
8    numStripes = dirU(data);//primera fase
9    for (i = 0; i < numStripes; i++) {
10     dirV(data, i); //segunda fase
11   }
12 }
```

Fig. 6: Conversión `Naive` de superficies NURBS a superficies Bézier.

anterior. HLSL 5.0 permite un máximo de 48 KB de memoria compartida por cada grupo de *threads*, sin embargo, por la naturaleza heterogénea de las superficies NURBS, es posible encontrar superficies de grandes dimensiones o con una gran cantidad de *knot spans*, de manera que al realizar su descomposición en la dirección v se generen una cantidad de *stripes* que no pueden ser almacenadas en la memoria compartida. Para solucionar este problema se usó un array de texturas que almacenará consecutivamente las *stripes* generadas. Además, se invoca el *pipeline* gráfico mediante la llamada `Inderect Drawing`, de esta manera permite que la entrada del `Vertex Shader` sea el propio *buffer* de salida del `Compute Shader`, evitando así la necesidad de realizar una copia de los datos en CPU. Por último, esta propuesta busca aumentar la ocupación de la

TABLA I: Controladoras gráficas utilizadas en las pruebas.

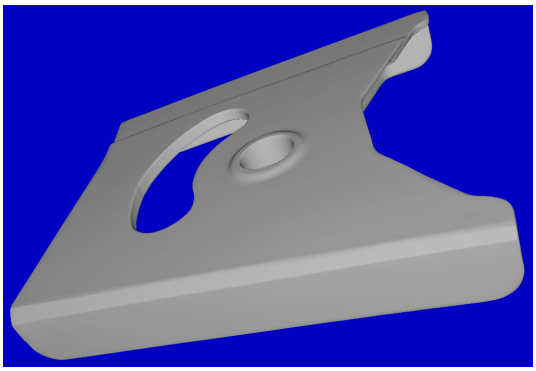
Modelo	Arquitectura	Memoria
NVidia GTX 1080ti	Pascal	11GB GDDR5X
AMD Radeon RX Vega 64	Vega	8GB HBM2

GPU dividiendo el trabajo de la segunda fase del algoritmo, la conversión en la dirección v, en diferentes *threads* dentro de cada grupo. Cada grupo se sigue encargando de la transformación de una única superficie NURBS, pero cuando se realiza la conversión en la dirección v, cada *thread* del grupo procesará un *stripe* diferente. El número de *threads* que se ejecuta en paralelo, `MAX_THREADS`, viene definido por el máximo número de *threads* ejecutables de manera simultánea permitidos por la arquitectura de la tarjeta gráfica.

La última propuesta, denominada `MultipleSurfaces`, aumenta el grado de paralelismo para aprovechar más adecuadamente la GPU. En este caso, se asignan varias superficies NURBS a cada grupo de *threads*. Se obtiene el número de *stripes* que generará cada superficie NURBS en la etapa de prepocesamiento de lectura del modelo. Con esta información se le asigna a cada grupo de *threads* tantas superficies NURBS como la suma de sus *stripes* sea menor a `MAX_THREADS`, que viene dado por la arquitectura de la GPU. Así, al realizar la conversión en la dirección u, cada grupo podrá calcular varias superficies NURBS simultáneamente. Una vez obtenidos todos los *stripes* del grupo, se realizará la conversión de los mismos en la dirección v asignado un *thread* a cada uno de los *stripes* obtenidos.

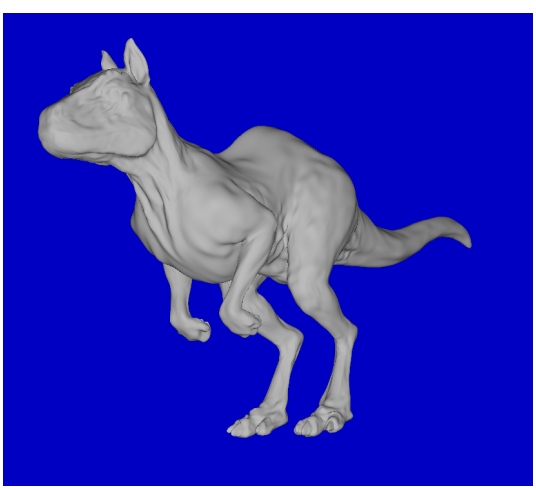
IV. Resultados

En esta sección se presentan los resultados para las diferentes propuestas expuestas en este trabajo obtenidos sobre diferentes modelos. La plataforma sobre la que se realizaron las diferentes pruebas consta de un Intel i7-4790 a 3,6 GHz con 32 GB de ram. Los dispositivos gráficos utilizados se pueden ver en la Tabla I. En cuanto al software, las pruebas se han realizado sobre Windows 10 usando Visual Studio Community 2017 con Direct3D 11 y Microsoft HLSL 5.0.



(a) Hinge

(b) Head



(c) Killero

Fig. 7: Modelos NURBS renderizados por nuestras propuestas.

TABLA II: Resultados modelos NVidia 1080ti.

Modelo	Superfi. NURBS	Número Stripes	Superfi. Bézier
Head	601	3005	15.025
Hinge	426	3822	34.842
Killeroo	89	627	11.532

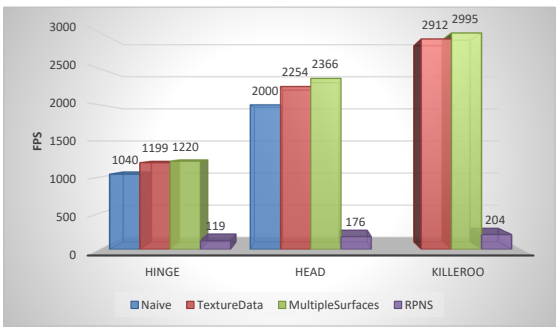


Fig. 8: FPS - Nvidia 1080Ti.

Los modelos usados para mostrar los diferentes resultados se pueden ver en la Figura 7, *Hinge* (Figura 7a), *Head* (Figura 7b) y *Killeroo* (Figura 7c). La Tabla II muestra para los tres modelos las superficies NURBS del modelo inicial, el número de *stripes* después de aplicar la primera fase del algoritmo y el número de superficies Bézier finales. Como se puede ver en la tabla, aunque el modelo *Head* tiene un mayor número de superficies NURBS que el modelo *Hinge*, es este último el que se descompone en un mayor número de Superficies Bézier.

La Figura 8 y la Figura 9 muestran el número de *frames* por segundo (FPS) medio para una resolución de 2048×1152 píxeles para los tres modelos, con las diferentes propuestas presentadas en este trabajo y dos arquitecturas diferentes de GPU. Estos valores se comparan con los obtenidos por RPNS en las mismas condiciones. Antes de entrar a analizar los resultados destacar una de las carencias de la versión `Naive`. En este caso, donde se usa la memoria compartida por grupo, el tamaño máximo de esta es de 48 KB. Las superficies NURBS que componen el modelo *Killeroo* son de grandes dimensiones, teniendo su mayor superficie 46 *knot spans* en la dirección u y 71 puntos de control en la dirección v, haciendo que los requerimientos de memoria superen los 100 KB. Por este motivo los resultados obtenidos empleando el modelo *Killeroo* con la versión mencionada no se muestran en las Figuras 8 y 9, debido a que su visualización no es correcta.

La Figura 8 muestra el rendimiento en la NVidia 1080Ti, mientras que la Figura 9 muestra el rendimiento en la AMD Vega 64. En ambos casos se ve un comportamiento similar. El modelo que

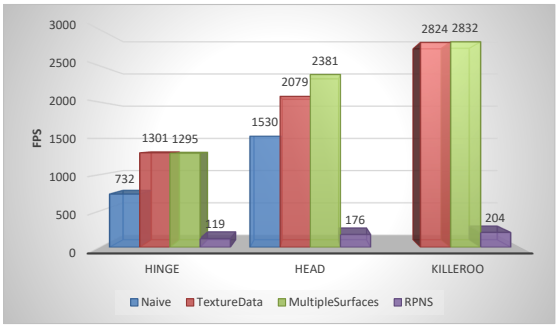


Fig. 9: FPS - AMD Vega 64.

obtiene una menor tasa de FPS es *Hinge*, este comportamiento se debe a que es el modelo que tiene un mayor número de superficies Bézier finales, con el coste computacional que ello conlleva. Por el contrario, el modelo con un menor número de superficies Bézier, el *Killero*, obtiene un mejor rendimiento en todos los casos. Comparando las diferentes propuestas entre sí, se aprecia como cada nueva propuesta consigue una mejora sobre la anterior. En la comparación de nuestros resultados en ambas GPUs con RPNS se ve que nuestras propuestas consiguen un mejor rendimiento que esta. Esto se debe a que RPNS esta haciendo el cálculo necesario para mostrar las superficies NURBS en todas las llamadas del *pipeline* gráfico. Por último, comparando el rendimiento de las 2 GPUs entre sí, se aprecia como es la Nvidia 1080Ti la que obtiene un mejor rendimiento.

V. Conclusiones

En este trabajo se presenta una nueva aproximación para realizar el *rendering* de modelos NURBS mediante la transformación de las superficies NURBS a superficies Bézier directamente en la GPU. De esta manera, se consigue aprovechar la capacidad de las tarjetas gráficas para realizar la conversión en la GPU usando del `Compute Shader`, evitando así posibles cuellos de botella en el envío de datos entre CPU y GPU. Se han mostrado una serie de propuestas, que van desde una implementación sencilla a una más especializada, que obtienen un muy buen rendimiento, destacando incluso ante otras propuestas del ámbito y quedando así patente que esta aproximación es una buena solución para el problema expuesto.

Agradecimientos

El presente trabajo ha sido financiado por el Ministerio de Economía y Competitividad de España y los fondos FEDER (80%), TIN2016-75845-P, por la Xunta de Galicia, cofinanciada con fondos FEDER en el marco del Programa de Consolidación de Grupos de Referencia Competitivos, ED431C 2017/04

y por el Centro Singular de Investigación de Galicia (acreditación 2016-2019) y la Unión Europea (Fondo Europeo de Desarrollo Regional, FEDER), ED431G/01.

Referencias

[1] L. Piegl, W. Tiller, *The NURBS Book*, 2nd Edition, Springer, 1997.

[2] E. Cohen, T. Lyche, R. Riesenfeld, *Discrete B-splines and subdivision techniques in computer-aided geometric design and computer graphics*, Computer Graphics and Image Processing, vol. 14, pp: 87–111, 1980.

[3] J. Shen, J. Kosinka, M. A. Sabin, N. A. Dodgson, *Conversion of trimmed nurbs surfaces to Catmull-Clark subdivision surfaces*, Aided Geometric Design, vol. 13, no. 7–8, pp: 486–498, 2014.

[4] R. Concheiro, M. Amor, M. Bóo, *Synthesis of Bézier Surfaces on the GPU*, en las actas del GRAPP'10: International Conference on Computer Graphics Theory and Applications, pp: 110–115, INSTICC Press, 2010.

[5] Y. I. Yeo, L. Bin, J. Peters, *Efficient Pixel-Accurate Rendering of Curved Surfaces*, en las actas del i3D'12: ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games, pp: 165–174, ACM, 2012.

[6] F. Claux, L. Barthe, D. Vanderhaeghe, J.-P. Jessel, M. Paulin, *Crack-free rendering of dynamically tesselated b-rep models*, Computer Graphics Forum, vol. 33, no. 2, pp: 263–272, 2014.

[7] M. Niessner, C. Loop, M. Meyer, T. DeRosse, *Feature adaptive gpu rendering of Catmull-Clark subdivision surfaces*, ACM Transactions on Graphics, vol. 31, no. 1, 2012.

[8] A. Krishnamurthy, R. Khardekar, S. McMains, *Direct Evaluation of NURBS Curves and Surfaces on the GPU*, en las actas del SPM'07: The 2007 ACM Symposium on Solid and Physical Modeling, pp: 329–334, 2007.

[9] A. Krishnamurthy, R. Khardekar, S. McMains, K. Haller, G. Elber, *Performing efficient nurbs modeling operation on the gpu*, Transactions on Visualization and Computer Graphics, vol. 15, no. 4, pp: 530–543, 2009.

[10] R. Concheiro, M. Amor, E. J. Padrón, M. C. Doggett, *Interactive rendering of NURBS surfaces*, Computer-Aided Design, vol. 56, pp: 34–44, 2014.

[11] Y. I. Yeo, L. Bin, J. Peters, *GPU-based Trimming and Tessellation of NURBS and T-Spline Surfaces*, en las actas del ACM SIGGRAPH'05, pp: 1016–1023, ACM, 2012.

[12] C. de Boor, *A Practical Guide to Splines*, Springer-Verlag, 1978.

[13] J. Zink, M. Pettineo, J. Hoxley, *Practical Rendering and Computation with Direct3D 11.*, Taylor & Francis, 2011.

Aceleración de Time-Series sismográficas en Python

Francisco López[1], Thomas Grass[1], Rafael Asenjo[2], Angeles Navarro[2]

Resumen— **Python se ha convertido en un lenguaje de programación muy popular, pero también es uno de los menos eficientes en términos de prestaciones y consumo energético. Este artículo describe el proceso que hemos seguido para acelerar una aplicación Python de tratamiento masivo de datos orientada a las Time-Series sismográficas, de manera que al usuario final se le sigue ofreciendo la productiva interfaz Python que tanta aceptación tiene. Este proceso se ha desplegado siguiendo una estrategia en tres fases. En la primera fase se ha aplicado un cambio algorítmico cuyo objetivo ha sido reducir la complejidad computacional del principal kernel (hot-spot) del código: las correlaciones cruzadas. Para ello se ha optado por implementar dichas correlaciones aplicando el Teorema de la Convolución. En la segunda fase se ha aplicado un cambio de modelo de programación que ha consisitido en la implementación en C++ del kernel, lo que nos ha permitido la utilización de la muy optimizada biblioteca FFTW. En la tercera fase, gracias al cambio del modelo de programación, aplicamos optimizaciones conscientes de la arquitectura, entre ellas OpenMP, para aprovechar los nodos multicore de nuestro sistema, o ArrayFire que nos permite hacer uso de aceleradores gráficos (con soporte en CUDA y OpenCL). Tras este proceso de optimización hemos obtenido una aceleración de 6121x sobre la aplicación original de partida.**

Palabras clave— **Time-series, procesado de señal, computación de altas prestaciones, OpenMP, ArrayFire, FFTW.**

I. Introducción

LAS aplicaciones de tratamiento masivo de datos están a la orden del día. La gran cantidad de sensores y otros dispositivos, como *smartphones* y componentes IoT, registran múltiples magnitudes que varían a lo largo del tiempo. Dentro de este marco, surge el término de Time-Series, que consisten en señales discretas que representan los distintos valores que una magnitud toma con el transcurso del tiempo. Por ello, invertir tiempo y esfuerzo, no sólo en el desarrollo de aplicaciones que extraigan información relevante, sino también en optimizar las ya existentes, se erige como una necesidad imperiosa por varios motivos.

El primer motivo es el tiempo que el usuario de estas aplicaciones tiene que esperar hasta que obtiene resultados con los que poder trabajar. Cuanto menos tarden nuestras aplicaciones en obtener los mismos resultados a partir de un mismo conjunto de datos, más comparaciones, estudios y análisis se podrán realizar. El segundo motivo es la energía consumida por los equipos que realizan el cómputo. En términos generales, cuanto menos tiempo tarde en completarse nuestra ejecución, menos energía tendremos que invertir. El tercer motivo es poder hacer frente a conjuntos de datos de tamaños mayores, muchas veces necesarios para poder aumentar la precisión de los resultados. Este motivo está relacionado con el primero, pero no siempre es únicamente una cuestión de tiempo, sino también de otros recursos, como memoria.

Por todos estos motivos, nos decidimos por optimizar una aplicación que trabaja con Time-Series sismográficas, aplicando conocimientos de procesado de señal. La estrategia de optimización que ilustramos en este trabajo es perfectamente extrapolable a Time-Series que representen cualquier otro tipo de magnitudes, pues nuestra aproximación es independiente del dominio de aplicación.

El problema en cuestión consiste principalmente en realizar correlaciones cruzadas, que constituyen el núcleo computacional del principal kernel del código. Para optimizar este kernel se ha utilizado una estrategia en tres fases. En la primera fase se ha hecho uso del Teorema de la Convolución [1], que nos ha permitido reducir la complejidad computacional del kernel mediante un cambio de algoritmo. En la segunda fase se ha aplicado un cambio de modelo de programación que ha consisitido en la implementación en C++ de dicho kernel. El cambio de modelo de programación nos ha permitido explotar tres factores:

i Reducir la sobrecarga que supone en Python la ejecución interpretada y la resolución de tipos de datos en tiempo de ejecución.
ii Habilitar la invocación de librerías optimizadas para ciertos dominios de aplicación, en nuestro caso hemos podido explotar la muy optimizada biblioteca FFTW [2]. Aunque invocamos esta librería en C++, desarrollaremos un interfaz que hará posible al usuario seguir trabajando desde Python en todo momento.
iii Habilitar el uso de frameworks de optimización conscientes de la arquitectura.

Precisamente, en la tercera fase de nuestra estrategia, gracias al cambio del modelo de programación, aplicamos optimizaciones conscientes de la arquitectura, entre ellas OpenMP [3] - para aprovechar los nodos multicore de nuestro sistema - o ArrayFire [4] - que nos permite hacer uso de aceleradores gráficos (con soporte en CUDA y OpenCL)-.

La estructura del artículo es la siguiente: en la Sección II se explicarán en profundidad las peculiaridades del problema y se refrescarán los conocimientos sobre el Teorema de la Convolución. A continuación,

[1]RWTH Aachen University, Institute for Communication Technologies and Embedded Systems. e-mail: {francisco.lopez, thomas.grass}@ice.rwth-aachen.de.
[2]Universidad de Málaga, Andalucía Tech, Depto. de Arquitectura de Computadores. e-mail: {asenjo,angeles}@ac.uma.es.

la Sección III describe como aplicamos la metodología en tres fases, mencionada previamente. Para cada una de las fases e implementaciones presentadas se incluye la evaluación experimental sobre la mejora que se consigue respecto a la versión Python original. Finalmente, la Sección IV resume algunos trabajos relacionados y la Sección V expone las conclusiones y las líneas de trabajo futuro.

II. Situación de partida

A. Visión general

El principal propósito del trabajo realizado es acelerar, en la medida de lo posible, el rendimiento de una aplicación de tratamiento masivo de datos. La versión original del código fue desarrollada por el personal del *School of GeoSciences*, de la *University of Edinburgh*, para tratar datos recogidos por sensores de actividad sísmica en las Islas Galápagos. Dicha versión original, que estaba escrita única y exclusivamente en Python, consiste en varias etapas de procesamiento; sin embargo, la mayor parte del tiempo de ejecución corresponde a la primera fase, en la que se han concentrado nuestros esfuerzos.

Esta primera etapa consiste en tomar los datos procedentes de los sensores, que han sido almacenados previamente, y realizar correlaciones cruzadas dos a dos. De cada una de estas señales de correlación se extraen los elementos máximos y mínimos, así como los desplazamientos para los que se producen, dentro de un rango que se puede introducir como parámetro de entrada. De este modo, los resultados que se extraen, originalmente, son cuatro matrices donde se pueden encontrar los máximos, mínimos y desplazamientos necesarios para ambos casos, de la correlación cruzada de cualquier par de Time-Series de entrada. En la Figura 1 puede observarse que a partir de un conjunto de datos con un determinado número de Time-Series se extraen cuatro matrices cuadradas, con tantos elementos en cada fila y columna como Time-Series de entrada se tengan. También queda denotada la peculiaridad triangular de la estructura de datos de salida, así como la variabilidad en las longitudes de las Time-Series que conforman el conjunto de datos de entrada.

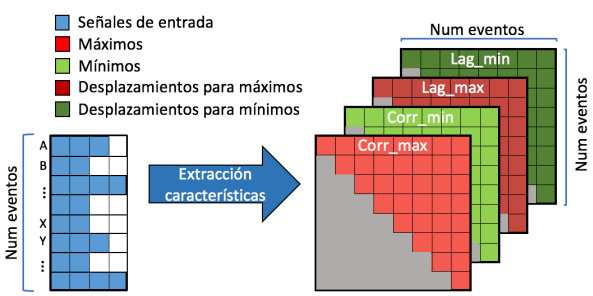


Fig. 1. Visión general del problema.

Una vez obtenidas estas matrices, se aplica un algoritmo de *clustering* para identificar familias de terremotos. Tanto la duración como el tipo de terremoto que conforman las familias ofrecen información sobre el sistema físico que los genera [5], [6].

El parámetro de entrada, *shift*, fija el rango en torno al centro de la señal de correlación cruzada donde se van a buscar los máximos y mínimos. Además, es reseñable indicar que se considera como centro de la correlación el valor resultado de alinear los elementos centrales de ambas Time-Series.

En la Figura 2 puede observarse un ejemplo de la operación de correlación cruzada que ha de realizarse entre cada par de Time-Series. Se observa que el *shift* toma un valor de tres, por lo que se buscarán los máximos y mínimos en tantos elementos alrededor del elemento central. Puede comprobarse que el elemento central es el resultado de realizar el producto escalar (*dot product*, en este caso igual a $X \cdot Y^T$) entre las dos Time-Series y que el resto de elementos se hallan desplazando una de las dos Time-Series. Así, los elementos que caigan fuera de la multiplicación se tomarán como nulos. En el ejemplo, cuando Y se desplaza se introducen ceros por el lado contrario hacia el que se realiza el desplazamiento. Todos los valores de CC son calculados con la siguiente expresión:

$$CC[i] = X[n] \cdot Y[n-i], i \in (-N, N)$$

donde N representa la longitud de las Time-Series, que suponemos del mismo número de elementos. Al contrario de lo que pueda considerarse en primera instancia, los valores finales de la correlación cruzada no dependen únicamente del número de multiplicaciones válidas sino también de los valores a multiplicar. Con esto, puede observarse que, en el ejemplo, el máximo no se halla en el elemento central, sino en el $lag\_max = 1$; por su parte, el mínimo se halla en $lag\_min = -1$. El máximo y mínimo correspondientes a estos desplazamientos son 117 y 58, respectivamente. Estos valores siempre cumplirán las ecuaciones siguientes:

$$CC[lag\_max] = max(CC[i]) = corr\_max$$

$$CC[lag\_min] = min(CC[i]) = corr\_min$$

Con todo lo mencionado previamente, se presenta ante nosotros un problema de carácter triangular, dada la simetría que desprenden las matrices de salida. Esto va a condicionar, en gran medida, la forma en que se implementarán las distintas versiones paralelas del código. No menos importante es el hecho de que al crear no sólo una, sino 4 matrices, donde la mitad de los elementos finales sabemos que serán ceros, estamos desperdiciando prácticamente la mitad de la memoria consumida por el programa. Este problema, como se tratará más adelante, ha sido abordado satisfactoriamente, de forma que únicamente se emplee la memoria que realmente sea necesaria.

En la Figura 3 se muestra el código Python de partida, que se irá optimizando en diferentes etapas, tal y como se describe en las siguientes secciones. En las líneas 2-7 se halla el número de eventos (Time-Series) y, en función de dicho valor, se crean las cuatro matrices de salida: `Corr_max`, `Corr_min`, `Lag_max`

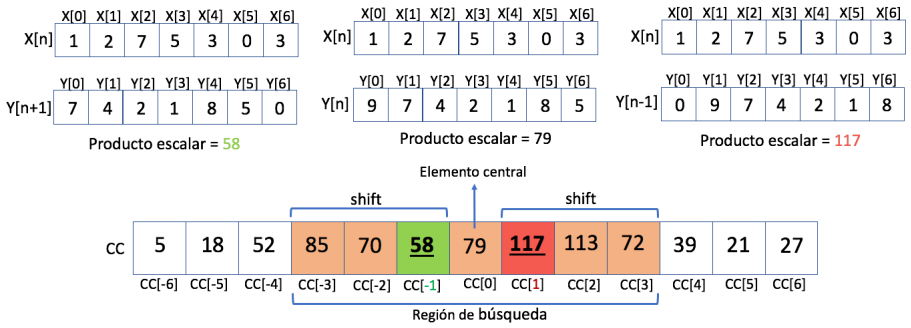


Fig. 2. Ejemplo de correlación cruzada de dos Time-Series

```
1  def XCM2(events):
2      n_events = len(events)
3
4      Corr_max = zeros((n_events,n_events))
5      Lag_max = zeros((n_events,n_events))
6      Corr_min = zeros((n_events,n_events))
7      Lag_min = zeros((n_events,n_events))
8
9      for i in range(n_events):
10         for j in range(i, n_events):
11             xcorrij = xcorr(events[i], events[j], 250, full_xcorr=True)
12             #Returns index and CC[i] for max(abs(CC[i])) --including negative values--
13             Lag_min[i,j] = xcorrij[0]
14             Corr_min[i,j] = xcorrij[1]
15             if xcorrij[1]<0.:
16                 #Return highest positive CC[i] and index (xcorrij[2] contains CC)
17                 Lag_max[i,j], Corr_max[i,j] = xcorr_max(xcorrij[2],abs_max=False)
18             else:
19                 Lag_max[i,j] = xcorrij[0]
20                 Corr_max[i,j] = xcorrij[1]
21
22     return Corr_max, Lag_max, Corr_min, Lag_min
```

Fig. 3. Código original en python.

y `Lag_min`, tal y como se ha explicado previamente. A continuación, comienza el doble bucle que recorre el triángulo superior de las matrices de salida. En la línea 11 se lanza el cómputo de la correlación cruzada correspondiente. Posteriormente, en las líneas 13-20 para cada señal de correlación se busca el máximo, el mínimo y los desplazamientos correspondientes. Sin embargo, es peculiar el hecho de que, si el valor absoluto del máximo es mayor que el valor absoluto del mínimo (que suele ser negativo), éste último tomará el valor del primero. Esto es así por motivos derivados del sentido físico de las señales sísmicas y las Time-Series que las representan.

B. *Teorema de la convolución*

El Teorema de la convolución supone un pilar fundamental sobre el que descansan, en gran medida, los avances conseguidos. Éste teorema afirma que la convolución puede realizarse en el dominio de la frecuencia como una simple multiplicación de las señales. Sin embargo, el propósito del código original no es computar convoluciones, sino correlaciones cruzadas, para lo cual se hará uso de la teoría de señales y sistemas lineales e invariantes en tiempo. Así, para obtener una convolución de dos Time-Series la cadena de operaciones consiste en realizar las transformadas de Fourier de cada Time-Series por separado, realizar la multiplicación elemento a elemento y, finalmente, computar la transformada de Fourier inversa del resultado. La única diferencia con la correlación cruzada estriba en tener que invertir el eje de tiempos de una de las señales antes de realizar la transformada de Fourier, tal y como se muestra en las ecuaciones siguientes, donde x e y denotan dos Time-Series de entrada:

$$Conv(x,y) = IFFT[FFT[x[n]] * FFT[y[n]]]$$

$$Cross{-}corr(x,y) = IFFT[FFT[x[n]] * FFT[y[-n]]]$$

No son pocas las implementaciones de algoritmos de comparación de Time-Series que hacen uso de esta propiedad; sin embargo, ha de tenerse en cuenta la longitud de las señales de entrada, es decir, el número de elementos que componen cada una de ellas. Así, para Time-Series compuestas por menos de un cierto número de elementos, no es beneficioso realizar la convolución en frecuencia. En nuestro caso, al pasar del dominio del tiempo al dominio de la frecuencia es posible reducir la complejidad del algoritmo de correlación cruzada de $\mathcal{O}(n^2)$ a $\mathcal{O}(n\log(n))$, pues, además, realizaremos un *padding* sobre las señales de entrada para que el número de elementos alcance la siguiente potencia de dos. Con este *padding* conseguiremos obtener una mejor eficiencia al calcular las FFT e IFFT.

C. *Setup experimental*

A lo largo del proyecto llevado a cabo se ha hecho uso del *clúster* MinoTauro [7] del Barcelona Super-

computing Center (BSC). Éste está compuesto por 38 servidores Bullx R421-E4, donde cada uno, a su vez, ofrece:

- CPU: 2 procesadores Intel(R) Xeon E5-2630 v3 @ 2,4 GHz con 20 MB de cache L3.
- GPU: 2 tarjetas NVIDIA(R) K80.
- 128 GB de memoria principal.
- *Peak Performance*: 250,94 TFlops
- Sistema operativo: RedHat Linux 6,7

Pra evaluar las distintas implementaciones, trabajaremos con un *dataset* formado por 6659 Time-Series, compuestas por 1501 puntos. Éste será el conjunto de datos de referencia. Originalmente, el tiempo de ejecución del código presentado en la Figura 3, con conjuntos de datos similares en tamaño, se ubica en 111774,44 segundos (31 horas, 2 minutos y 54 segundos), empleando el clúster MinoTauro.

Como se demostrará en la siguiente sección, el margen de mejora posible sobre el código original es bastante significativo. Varios motivos propician esta situación, tanto el hecho de realizar todas las correlaciones cruzadas en el dominio temporal como el emplear un lenguaje interpretado de alto nivel como es Python. Además, el problema es altamente paralelo (*embarrassingly parallel*), pues todos los elementos de las matrices de salida a calcular son independientes entre ellos. Así, cada uno de los elementos de las matrices de salida únicamente compartirá una de las señales de entrada con los elementos que se encuentren en la misma fila o columna. Esta propiedad la podremos considerar cuando apliquemos optimizaciones conscientes de la arquitectura.

III. Mejoras aplicadas

En las siguientes subsecciones se expondrán las sucesivas mejoras que se han implementado. Sin embargo, es importante tener en mente que todas ellas comparten una interfaz desde Python, pues el objetivo es que el problema pueda seguir computándose con una simple llamada a una función desde la terminal, pero en un tiempo mucho menor. De aquí en adelante todos los tiempos mostrados serán el resultado de ejecutar la computación en el servidor MinoTauro del BSC, en aras de poder realizar una justa comparación de los tiempos obtenidos.

A. Fase 1: Optimización algorítmica. Paso al dominio de la frecuencia

El código inicial hace uso de funciones de *ObsPy* [8], que se ha establecido como un *framework* de referencia para procesamiento de datos sismológicos. Su objetivo principal es facilitar un desarrollo cómodo y sencillo de aplicaciones de este ámbito.

Tal y como se ha explicado previamente, se puede conseguir una gran mejora en el tiempo de computación si las correlaciones se realizan en el dominio de la frecuencia, pues reduciremos la complejidad computacional pasando de $\mathcal{O}(n^2)$ a $\mathcal{O}(n\log(n))$. Por ello, se comenzará por emplear una función que pertenece a una versión más reciente de la propia librería *Obspy*: *correlate*. Esta función realiza una llamada a la función *fftconvolve* de *SciPy*, si se va a computar en el dominio de la frecuencia. Sin embargo, el dominio en el que queremos que se realice la correlación puede ser fijado de antemano; dado que a partir de señales de 100 elementos se fija el umbral para que sea beneficioso trabajar en el dominio de la frecuencia. Nosotros especificaremos que sea éste el dominio en el que queremos que se compute.

Con este simple cambio ya se obtiene una primera mejora evidente respecto a la versión original. El tiempo de ejecución es de 125,62 minutos, frente a las más de 31 horas de la versión original (speedup de 14,83x). En esta ocasión no se está explotando toda el potencial del Teorema de la Convolución porque las Time-Series con las que se está trabajando no tienen un número de elementos potencia de dos. Aún así, para haber realizado un cambio tan sutil como sustituir una llamada de una función por otra, el resultado es más que esperanzador.

B. Fase 2: Optimización del modelo de progrmación

B.1 Compilación con *Cython*

El siguiente paso para reducir el tiempo de ejecución del código Python consiste en evitar la ejecución interpretada y la resolución de los tipos de datos en tiempo de ejecución (tipado dinámico). Existen distintas alternativas para conseguir ese objetivo, pero la más inmediata se apoya en usar compiladores fuente-fuente. En este trabajo, hemos usado Cython que es a su vez un lenguaje de programación que extiende a Python y un compilador de Cython a C/C++ que elimina gran parte del overhead de ejecución de Python.

En su vertiente como lenguaje, Cython permite especificar el tipo de las variables de un programa Python (int, double, etc). Esto permite eliminar casi en su totalidad el tiempo que Python tiene que dedicar a consultar el tipo de datos de cada variable que tiene que leer o escribir. En su vertiente como compilador fuente-fuente, la traducción de Python a C/C++, y la posterior compilación del código C/C++ para generar una librería dinámica, elimina el overhead de interpretación de código Python.

Estas dos ventajas (reducción del tipado dinámico y eliminación del intérprete de Python) no llevan aparejadas una merma en la productividad del desarrollador ya que sólo se requiere modificar ligeramente el código Python para especificar el tipo de las variables y usar el tool-chain de Cython para generar la libería dinámica. Además, de cara al usuario final, esa librería se carga como un módulo Python convencional (desde cualquier código o intérprete Python) y las funciones Cython ya compiladas en la librería pueden ser invocadas como cualquier función Python. La única diferencia es que la ejecución es más rápida ya que la llamada a la función Cython está finalmente ejecutando código nativo de la librería dinámica.

Empleando Cython se consigue un tiempo de ejecución de 8381,21 segundos (2 horas, 19 minutos y 41

segundos) y, por tanto, un speedup de 13,34x sobre el *baseline*. Vemos que en este código, la implementación en Cython no mejora a la versión que llama a la función *correlate* de la librería *Obspy*. Aunque no hemos podido confirmarlo experimentalmente, pensamos que, dado que *correlate* ya está implementado en C dentro de la librería *Obspy* y que la llamada a esta función es la que más tiempo consume, el uso de Cython no está consiguiendo una implementación mejor que la que ya incluye la librería *Obspy*.

B.2 Ctypes y C++

Dado que nuestro objetivo es conseguir ejecutar el problema en el menor tiempo posible, y tras evidenciar la capacidad de aceleración en la ejecución al computar las correlaciones en el dominio frecuencial, el siguiente paso es crear una versión en un lenguaje completamente compilado que explote esta característica. Aunque la función *correlate* de la librería *Obspy* ya está implementada en C, nos aventuramos a realizar nuestra propia implementación con la esperanza de conseguir aún mejores resultados. Esto se justifica porque no sólo desarrollaremos nuestra implementación de la correlación cruzada (línea 11 en la Figura 3) sino que ya de paso implementamos en C++ todo el bucle externo (línea 9 en la Figura 3) que se encarga de hacer todas las comparaciones dos a dos de las Time-Series.

Con este objetivo se ha empleado la biblioteca FFTW [9], cuyas siglas provienen de *Fastest Fourier Transform in the West*. Esta biblioteca, escrita en C, proporciona funciones para que el cómputo de las FFT e IFFT sea muy eficiente. El uso básico de esta biblioteca consta de dos pasos: primero se define el plan y posteriormente se ejecuta. En la creación del plan se indican tanto las posiciones de memoria donde comienzan los datos de entrada y los datos de salida de la operación, como el número de elementos de la operación a realizar. Además, en la creación del plan se pueden añadir flags que permiten indicar si se optimiza la ejecución de la FFT (o IFFT) o si se realiza una estimación de los parámetros, obteniendo una ejecución subóptima. Los flags, respectivamente, son *FFTW_MEASURE* y *FFTW_ESTIMATE*. Como cabe esperar, la obtención de parámetros óptimos no es gratis, sino que conlleva un mayor overhead inicial en la operación.

En esta versión básica se va a crear un plan para cada FFT e IFFT ejecutadas, por ello, el flag *FFTW_ESTIMATE* es el recomendado. En versiones posteriores, como se explicará, se creará un plan constante y únicamente variarán los datos sobre los que se aplica. En este caso será beneficioso usar el flag *FFTW_MEASURE*.

Como se ha mencionado previamente, se hará uso de una interfaz Python que mantenga la productividad del programador al tiempo que internamente se explote una implementación más eficiente. En la Figura 4 se pueden observar las distintas capas presentes al emplear el código desarrollado. El bloque `test_c.py` (Algorithm 1) representa el código que el usuario desarrollaría, donde se han de cargar los datos de entrada e importar el *Interfaz Python*. Este interfaz, a su vez, hace uso de las funciones implementadas en C++.

A continuación, se explicarán las peculiaridades de los dos componentes inferiores:

- Interfaz Python (`correlation_lib.py`, Algorithm 2): en primer lugar realiza la carga, con *ctypes*, de la librería dinámica producto de compilar el código en C++ (línea 1). Ctypes [10], es una librería que permite hacer llamadas a código C/C++ desde código Python de forma productiva y sin pérdida de rendimiento debida a movimiento de datos (siempre que se usen determinadas estructuras de datos linearizadas). Posteriormente, extrae los datos del objeto *obspy.core.trace.Trace*, en el que se encuentran las Time-Series en el problema original, y los escribe en un array bidimensional de forma que todas las Time-Series tengan la misma longitud, que será potencia de dos (líneas 2-4). Finalmente, se realiza la llamada a la función del código C++ (línea 5) que devuelve las matrices con los valores deseados de máximo, mínimo y desplazamientos.
- Código C++ con FFTW (`correlation_c.cpp`, Algorithm 3): en esta implementación se ha modificado la forma de realizar el algoritmo. Inicialmente, por cada elemento de las matrices de salida se realizaban dos FFTs, una multiplicación elemento a elemento y una IFFT, junto con la búsqueda de las características. En nuestro caso, hemos optado por calcular todas las FFTs al comienzo de la ejecución (línea 1), almacenarlas en memoria y realizar las lecturas necesarias para las multiplicaciones posteriores. Así, se ha reducido el número de FFTs a realizar, de $\mathcal{O}(n^2)$ a $\mathcal{O}(n)$. El siguiente paso es calcular las normas de todas las Time-Series (línea 2), pues queremos que los resultados de la correlación cruzada estén normalizados entre -1 y 1. Además, se reserva la memoria necesaria para guardar el resultado de la multiplicación elemento a elemento de dos señales en frecuencia. Finalmente, se da comienzo al doble bucle que itera sobre los distintos elementos de las matrices de salida, realiza los productos elemento a elemento pertinentes (línea 5) y, tras la IFFT (línea 6), extrae las características (línea 7).

Con todo lo mencionado anteriormente, empleando el *dataset* de referencia se obtiene un tiempo mediano de 25,09 minutos. Esto supone un speedup de 74,25x sobre el código original y de 5,57x sobre la versión de Cython. Se aprecia que se obtiene un importante speedup respecto a la versión de Cython, pero esto requiere más esfuerzo por parte del desarrollador.

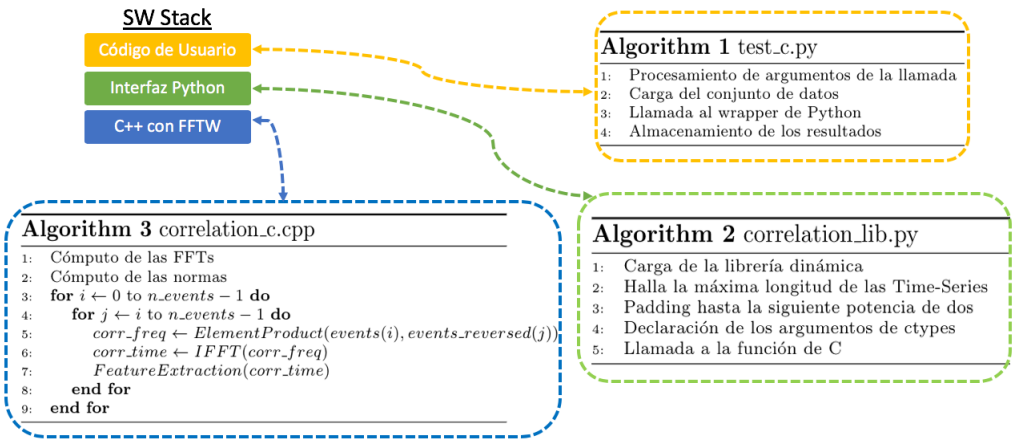


Fig. 4. Jerarquía de llamadas de la implementación en C++.

C. Fase 3: Optimizaciones conscientes de la arquitectura

C.1 Paralelización con OpenMP

Sabiendo que nos enfrentamos a un problema con un elevado grado de paralelismo, donde los elementos a computar son independientes entre sí, es imperioso emplear todos los núcleos del procesador para intentar acelerar la ejecución de la aplicación. Para ello, sobre la versión básica de C++, se van a integrar las llamadas necesarias para convertir nuestra versión serie en una versión paralela.

Como cabe esperar, la sección paralela comprende el doble bucle. Por tanto, como cada hilo va a computar multiplicaciones elemento a elemento e IFFTs independientemente, todos necesitan memoria donde almacenar estos resultados. Esta memoria será privada para cada thread.

En primer lugar, se empleará un *scheduler* estático, que repartirá el conjunto de iteraciones del bucle exterior de forma equitativa entre los hilos que especifique el usuario de la aplicación, desde Python. Para este problema en concreto, el *scheduler* estático no es particularmente beneficioso por el carácter triangular que presenta. Así, al primer hilo se le asignará un mayor número de iteraciones paralelas, que disminuirá progresivamente hasta el último hilo, para de esta manera garantizar que la carga de trabajo en cada hilo esta equidistribuída. Por tanto, podemos esperar que el tiempo total de la ejecución sea igual que el tiempo que tarde el primer hilo en terminar de computar la carga de trabajo que se le ha asignado.

Si se consideran los elementos de las matrices a calcular como una superficie, podemos determinar qué carga de trabajo paralelo (N_i) se asigna a cada hilo con la ecuación mostrada a continuación (empleando el planificador estático):

$$N_i = (n_{events}/n_{threads})^2 * (n_{threads} - (i + 1/2))$$

con i entre [0, $n_{threads}$). En esta ecuación n_{events} representa el número total de Time-Series de entrada y $n_{threads}$, el número de hilos totales. Por su parte, i representa el índice de cada hilo. Tal y como se ha mencionado previamente, puede observarse que número de iteraciones paralelas a computar por cada hilo disminuye al incrementar i.

En la Tabla I se muestra la estimación analítica del speedup máximo que puede esperarse al incrementar el número de hilos, empleando el planificador estático previamente mencionado. Estos valores han sido verificados experimentalmente, siendo los valores medios similares a los estimados analíticamente. Por ejemplo, para 8 hilos, se obtiene un tiempo de 360.3 segundos, que equivalen a poco más de 6 minutos (speedup de 310,23x sobre el baseline).

TABLA I
Speedup máximo con el planificador estático.

N. Proc.	1	2	4	8
Speedup	1	1,33	2,29	4,27

Claramente, con este planificador no se explota todo el paralelismo que ofrece el problema original y, por ello, se emplearán otros como el *Dynamic* o el *Guided*. Estos dos planificadores presentan un parámetro, *chunk size*, que permite fijar, en el primer caso, el tamaño máximo del número de iteraciones que se va asignando a cada hilo, mientras que en el segundo caso determina el tamaño mínimo del bloque asignado. Esto se debe a que el *Dynamic* va asignando bloques de iteraciones a los hilos según vayan pidiendo más carga de trabajo, mientras que el *Guided* realiza un reparto de forma proporcional entre el número de iteraciones que quedan por distribuir y el número total de hilos en ejecución. Este parámetro puede tener un impacto perjudicial en el tiempo de ejecución si el valor indicado es relativamente elevado respecto al número total de iteraciones del bucle exterior. Varios factores influyen en una correcta elección de este parámetro, tales como el tamaño de las Time-Series (para el uso de cachés), el número total de iteraciones y el número total de hilos. Para nuestro *dataset* de referencia, se recomienda un valor entre 10 y 200 para el *chunk size*.

En la Figura 5 se muestran los speedup obteni-

dos empleando tanto el planificador *Dynamic* como el *Guided* y el *Static*. Como cabría esperar, el *Dynamic* devuelve mejores resultados, pues reparte bloques del mismo tamaño durante toda la ejecución, a excepción de la parte final del bucle; mientras que el *Guided* distribuye bloques cuyo tamaño depende del número de iteraciones restantes. Además, podemos comprobar que los valores de speedup obtenidos para el planificador *Static* se aproximan mucho a los valores teóricos calculados previamente.

Además, a fin de comprobar todas las posibilidades que ofrece el problema, se ha estudiado emplear el planificador *Guided* recorriendo el conjunto de iteraciones de final al principio (en sentido inverso del bucle "i"). Así, al distribuir de forma conjunta más iteraciones inicialmente y menos al final, habrá un mayor balance en la carga entregada a los hilos.

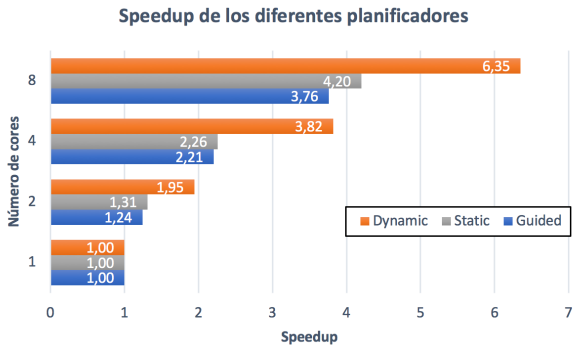


Fig. 5. Speedup obtenido con los distintos planificadores

En la Tabla II pueden observarse los tiempos de ejecución, para diferente número de hilos empleados. Se observa que, para el planificador *Guided* que recorre el conjunto de iteraciones de final a principio ("Guided rev.") se obtienen los mismos tiempos que para el planificador *Dynamic*.

TABLA II
Tiempos de ejecución para distintos planificadores y cores.

N. Proc.	1	2	4	8
Guided	1512,69	1216,60	685,89	402,25
Dynamic	1509,49	777,38	396,11	238,18
Guided rev.	1509,97	783,87	394,32	240,75

C.2 Optimizaciones avanzadas sobre OpenMP

En este punto, discutiendo con el personal de la *School of GeoSciences*, de la *University of Edinburgh*, llegamos a la conclusión de que no era necesario emplear los datos en formato de doble precisión, pues sólo los 3 primeros decimales son necesarios para la aplicación. Por lo tanto, se pueden declarar los arrays para que sean de tipo *floats*. Al cambiar el tipo de dato, además, cambian las estructuras de datos de la FFTW que se han de invocar (así como las llamadas a otras funciones auxiliares), que ahora operarán sobre datos de simple precisión. En la Tabla III se pueden observar los distintos valores medianos obtenidos al ir variando el número de hilos en ejecución cuando se aplica la optimizacion de simple precisión sobre la implementación OpenMP con planificador Dynamic. Claramente al pasar de de 4 a 8 núcleos no se está obteniendo una mejora, siquiera, considerable.

TABLA III
Tiempos de ejecución empleando datos con simple precisión para el planificacor Dynamic.

N. Proc.	1	2	4	8
Tiempo	823,62	429,52	235,75	229,33

Realizando un *profiling* sobre la aplicación desarrollada hasta ahora, se descubre que la mayor parte del tiempo de ejecución se debe a la creación y destrucción de los planes de las IFFTs. Por ello, se reescribió todo el código correspondiente a esta sección, declarando un único plan por cada hilo, que se reutilizará continuamente. Por tanto, en la computación de la IFFT ya no habrá presente una sección crítica, que previamente se empleaba para realizar la declaración de los planes, pues la única rutina *thread-safe* es la *fftwf_execute*, que lanza la computación de la FFT o IFFT en cuestión.

En la Tabla IV pueden observarse los valores obtenidos para esta nueva versión, que demuestran la importancia de declarar cuantos menos planes como sea posible. Además, con esta nueva versión, se optimiza la computación de la IFFT para estas longitudes de Time-Series, con el flag *FFTW_MEASURE*, del que ya se ha hablado previamente. Asimismo, la escalabilidad con el número de hilos es prácticamente ideal, excepto alguna pequeña desviación numérica.

TABLA IV
Tiempos de ejecución reescribiendo el proceso de IFFT.

N. Proc.	1	2	4	8
Tiempo	482,02	241,40	121,39	61,07

De momento, se habría obtenido con 8 núcleos un speedup muy similar al de ArrayFire (ver Sección III-C.3). Sin embargo, la versión de CPU seguía empleando datos complejos como entrada y, estudiando el manual de FFTW concienzudamente, pudimos encontrar las interfaces avanzadas que permiten realizar operaciones FFTs e IFFTs entre reales y complejos directamente. Esto permitió reducir, además, el tamaño que ocupaban las Time-Series, pues antes realizábamos un *padding* intercalado, para tomar la parte real proveniente de los datos y fijar las partes imaginarias a cero. Ahora, únicamente hay que realizar un *padding* hasta la siguiente potencia de dos.

Con todo esto, las señales en dominio de la frecuencia ocupan $N/2+1$ elementos complejos (donde N es la longitud de la Time-Series en dominio temporal), donde únicamente están presentes la mitad de los coeficientes, pues la otra mitad no es más que la conjugada de la primera. Finalmente, en la Tabla V puede observarse el efecto de saber explotar toda la capacidad que la biblioteca FFTW ofrece. Sigue ob-

servándose la cuasi-perfecta escalabilidad que presentan los resultados obtenidos. Con todos los cambios realizados, para 8 cores se obtiene un speedup de 5990x sobre el código original.

TABLA V
Tiempos de ejecución con uso de funciones de real a complejos y viceversa.

N. Proc.	1	2	4	8
Tiempo	139,42	69,72	35,65	18,66

Adicionalmente, algunas mejoras añadidas han sido aplicadas:

- Se ha paralelizado el cálculo inicial de las FFTs.
- Se ha modificado la estructura de datos que componen las matrices de salida para que únicamente almacenen los valores necesarios, sin tener que reservar memoria para la triangular inferior de cada una.
- Se han implementado funcionalidades que originalmente el personal de la *University of Edinburgh* realizaba en una segunda fase, como son el cálculo de un histograma de las magnitudes y la aplicación de un filtrado, que elimina valores por debajo de un cierto umbral. Tanto el número de *bins* del histograma como el valor para el filtrado son introducidos por el usuario desde Python.
- Los resultados se almacenarán como matrices dispersas comprimidas, permitiendo ahorrar gran cantidad de espacio (especialmente tras el filtrado).

Con todo lo comentado previamente, la versión final del código presenta unos tiempos de ejecución que podemos ver en la Tabla VI. Estos tiempos sólo mejoran ligeramente los tiempos para 4 y 8 núcleos cuando los comparamos con los obtenidos en la Tabla V.

TABLA VI
Tiempos de ejecución con funcionalidades añadidas.

N. Proc.	1	2	4	8
Tiempo	141,83	70,19	35,56	18,26

Además, se ha realizado la implementación de una versión que incluye *caché blocking* y de una versión que realiza múltiples IFFTs en paralelo. Sin embargo, ninguna de las versiones consigue mejorar el tiempo de ejecución obtenido anteriormente. La versión que explota *caché blocking* no consigue mejorar los resultados porque la forma de acceder los datos ya explota las localidades espacial (el *pre-fetching* se encarga de proporcionar lo datos necesarios satisfactoriamente) y temporal.

La principal motivación de hacer múltiples IFFTs en paralelo vino dada por el resultado obtenido al realizar un *profiling* sobre las iteraciones del bucle. Se obtuvo un tiempo de 6,4 microsegundos por elemento de salida, cuyo porcentaje de tiempo invertido

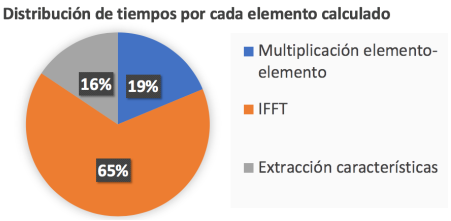


Fig. 6. Análisis detallado del tiempo por Time-Series

en cada operación puede observarse en la Figura 6. Claramente, la mayor parte del tiempo se dedica a realizar IFFTs. Sin embargo, el cómputo de múltiples IFFTs a la vez no devolvió mejores resultados.

C.3 Implementación con ArrayFire

Hasta ahora únicamente hemos explotado la potencia computacional de los cores incluidos en la CPU. Sin embargo, dado que nuestro problema en cuestión exhibe un elevado grado de paralelismo y que la plataforma de ejecución, MinoTauro, también contiene GPUs, cobra sentido usar también estos aceleradores. El objetivo es proporcionar aún menores tiempos de ejecución y explotar toda la potencia computacional disponible. Aunque las 2 tarjetas GPU disponibles por cada nodo son NVIDIA, en lugar de emplear CUDA, únicamente válido para este tipo de procesadores gráficos, se ha decidido realizar una versión portable a cualquier plataforma que soporte CUDA u OpenCL.

Para ello, trabajamos con ArrayFire [11], librería *Open Source* que facilita el desarrollo de aplicaciones tanto para arquitecturas paralelas (CPU) como masivamente paralelas (GPU). ArrayFire proporciona al desarrollador una abstracción de alto nivel, que permite a éste emplear los formatos de datos y funciones definidas por la librería. El código desarrollado se traduce en *kernels* que se ejecutan en diferentes plataformas, como CPUs de Intel, AMD, Arm y GPUs de NVIDIA, AMD y Qualcomm. Un aspecto relevante de ArrayFire es que cada llamada a una función del API de ArrayFire no se traduce en la invocación de un kernel a la GPU. En su lugar, un procesado Just In Time, JIT, en tiempo de ejecución, combina un número configurable de llamadas consecutivas al API en un único kernel. Esto resulta en menos llamadas al driver de GPU y kernels de mayor granularidad que consiguen reducir los *overheads* debidos a sincronizaciones entre el *host* y el dispositivo.

En la Figura 7 se puede observar el código en C++ que emplea ArrayFire. En primer lugar, entre las líneas 1-13, se observan varias llamadas al constructor de *af::array*, que crea arrays en el dispositivo en que se trabaje. Los arrays creados se corresponden con las Time-Series de entrada y las mismas invertidas, las matrices de salida, las normas de las señales y varios arrays auxiliares para poder almacenar la correlación cruzada y los valores a extraer. Tras estas declaraciones, comienza el doble bucle, en la línea 15 que conforma la parte computacionalmente más pesada de la aplicación. En este doble bucle se reali-

```
1  af::array tss(event_length, n_events, events); //creates array in the device
2  af::array tss_flipped = af::flip(tss, 0);
3
4  af::array af_Corr_max = af::constant(0, tss.dims(1), tss.dims(1), af::dtype::f32);
5  af::array af_Corr_min = af::constant(0, tss.dims(1), tss.dims(1), af::dtype::f32);
6  af::array af_Lag_max = af::constant(0, tss.dims(1), tss.dims(1), af::dtype::s32);
7  af::array af_Lag_min = af::constant(0, tss.dims(1), tss.dims(1), af::dtype::s32);
8
9  af::array norms = af::matmul(matrixNorm(tss, 0).T(), matrixNorm(tss, 0));
10
11 af::array corr = af::constant(0, tss.dims(0), tss.dims(1), tss.type());
12 af::array magnitude;
13 af::array lag;
14
15 for(int row=0; row<n_events; row++){
16       corr = af::constant(0, tss.dims(0), tss.dims(1), tss.type());
17
18       gfor(af::seq column, row, n_events-1){
19         corr(af::span, column) = af::convolve(tss(af::span, row), tss_flipped(af::span, column, 0),
                 AF_CONV_DEFAULT, AF_CONV_FREQ);
20       }
21
22       af::max(magnitude, lag, corr.rows(event_length/2 - shift, event_length/2 + shift), 0);
23       af_Corr_max(row, af::span) = magnitude;
24       af_Lag_max(row, af::span) = lag;
25
26       af::min(magnitude, lag, corr.rows(event_length/2 - shift, event_length/2 + shift), 0);
27       af_Corr_min(row, af::span) = magnitude;
28       af_Lag_min(row, af::span) = lag;
29 }
```

Fig. 7. Código desarrollado en C++ con ArrayFire.

za una llamada a la estructura de control *gfor*, en la línea 18 la cual permite lanzar muchas iteraciones del bucle de forma simultánea, gracias a la replicación de datos (*tiling* según ArrayFire). Así, en la línea 19, se realiza la llamada a la función de ArrayFire que computa la correlación cruzada. Posteriormente, en las líneas 22-28 se observa la extracción de características, primero para el máximo y su desplazamiento y, después para el mínimo y su desplazamiento correspondiente.

Una vez ha terminado la ejecución del doble bucle, realizamos un ajuste de los valores obtenidos, dividiendo las magnitudes entre la norma correspondiente y corrigiendo los desplazamientos. Finalmente, se copian los datos de vuelta al *host*.

Con el código desarrollado, haciendo uso del *backend* de CUDA, se obtiene un tiempo mediano de 131,56 segundos, que supone un speedup de 849,61x sobre el código original. En este punto, tras aplicar la recomendación de que no es necesario emplear los datos en formato de doble precisión, pues sólo los 3 primeros decimales son necesarios para esta aplicaciones, se decide reescribir el código para que todos los arrays fuesen de *floats*, es decir, simple precisión. Esta optimización tiene gran relevancia, especialmente en las arquitecturas de NVIDIA, donde suele haber el doble de unidades de procesamiento de datos de simple precisión. Con ello, obtuvimos un tiempo de ejecución de 64,48 segundos, lo que supone un speedup de 1733,50x respecto al problema original.

D. *Resumen de los resultados de las distintas implementaciones*

En la Figura 8 pueden compararse, en escala logarítmica, los speedups obtenidos (desde 1 a 8 cores) para cada implementación de las optimizaciones que se han aplicado en la estrategia de 3 fases que se ha seguido en este trabajo.

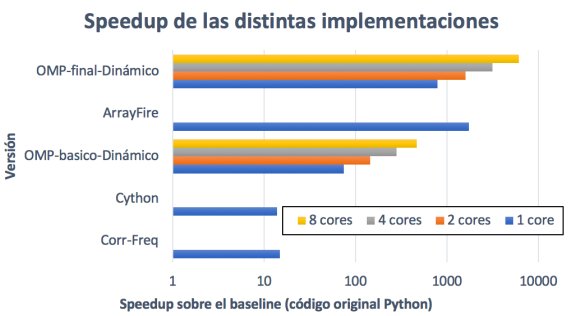


Fig. 8. Comparación de los speedups obtenidos por versión

Durante la primera fase, con la aplicación del Teorema de la Convolución para realizar correlaciones de forma más eficiente en el dominio de la frecuencia, conseguimos reducir la complejidad computacional. Para ello se invocan las funciones *fftconvolve* de la librería *SciPy* de Python, obteniendo la primera versión del código que denominamos `Cor-Freq`, la cual consigue una aceleración de 14,83x con respecto a la versión original. Durante la segunda fase, en la que apostamos por un cambio en el modelo de programación, primero tratamos, usando Cython que incorpora un compilador a C/C++, eliminar gran parte del overhead de ejecución de Python debido a la ejecución interpretada y la resolución de los tipos de datos en tiempo de ejecución. Esta nueva implementación es la que denominamos `Cython` en la figura. A continuación realizamos una implementación nativa en C++ que invoca funciones de la librería FFTW, y donde se puede observar el gran beneficio al desarrollar un código en un lenguaje con tipado estático y emplear librerías muy optimizadas para el cálculo de FFTs. Para esta versión ya tenemos una

speedup de 74,25x (ver OMP-basico-Dinámico para 1 core en la figura). El cambio de modelo de programación nos permite, en una tercera fase, añadir más optimizaciones conscientes de la arquitectura. Así realizamos una primera implementación paralela del bucle externo del kernel empleando OpenMP, y tras el estudio de distintas estrategias de planificación encontramos que *Dynamic* es la que mayor rendimiento proporciona, consiguiendo una speedup de hasta 469,29x (ver OMP-basico-Dinámico para 8 cores en la figura). Tras analizar la importancia de hacer un uso optimizado de las funciones de la librería FFTW y de declarar cuantos menos planes posibles para optimizar del uso de memoria, lo que conseguimos explotando localidad mediante privatización de buffers compartidos, a la vez que eliminamos puntos de sincronización debidos a secciones críticas, conseguimos reducir aún más los tiempos de computación, alcanzando ahora una speedup de 6121,3x con 8 cores (ver OMP-final-Dinámico para 8 cores en la figura). También estudiamos una nueva implementación con la librería ArrayFire lo que nos permite explotar la GPU de nuestro sistema, conseguiendo en este caso una speedup de 1733,5x (ver ArrayFire en la figura).

IV. Trabajos relacionados

Actualmente, muchos esfuerzos y proyectos están enfocados a mejorar las aplicaciones de tratamiento de Time-Series. Sin ir más lejos, la aplicación para *smartphones*, *Shazam* [12] está basada en el tratamiento de pequeños fragmentos de canciones capturadas con el terminal móvil, mediante la realización de transformadas de Fourier que permitan identificar los picos de frecuencias con los de canciones previamente almacenadas en una base de datos.

También encontramos empresas españolas dentro de este ámbito, como *Shapelets*, que ha desarrollado una librería que incluye técnicas novedosas de tratamiento de Time-Series, *Khiva* [13], empleando ArrayFire.

Por otra parte, también son dignas de mención las empresas como Ericsson o Huawei, que forman parte del 3GPP (*3rd Generation Partnership Project*), y actualmente están desarrollando algoritmos de extracción de información a partir de registros (Time-Series) para 5G. Estos desarrollos están enfocados no sólo a extraer la información, sino también a hacerlo lo más eficientemente posible.

V. Conclusiones y líneas de trabajo futuro

En este trabajo hemos ilustrado cómo utilizando una estrategia en tres fases, podemos acelerar en más de tres ódenes de magnitud una aplicación de Python que realiza tratamiento masivo de datos para el procesado de Time-Series sismográficas. Nuestra estrategia ha demostrado que elegir el algoritmo más apropiado para el tratamiento de la información es el primer paso para reducir la complejidad computacional, y por lo tanto el tiempo. En nuestro caso, pasar a trabajar en el dominio de la frecuencia ha supuesto un orden de mangnitud de mejora. A continuación, la selección del modelo de programación con el que se optimiza el núcleo computacional de la aplicación es otro paso crítico. En nuestro caso de estudio, el codificar en Ctypes y C++ nos ha permitido reducir la sobrecarga que supone en Python la ejecución interpretada y la resolución de tipos de datos en tiempo de ejecución, a la vez que nos abre la posibilidad de usar librerías optimizadas para ciertos dominios de aplicación, en particular en nuestra aplicación, la librería FFTW. Todo ello ha supuesto casi dos órdenes de magnitud de mejora. Por último, y gracias al cambio del modelo de programación hemos podido aplicar un conjunto de optimizaciones de bajo nivel conscientes de la arquitectura (paralelización basada en OpenMP, optimización de la localidad mediante privatización y eliminación de secciones críticas, y aceleración en GPU basada en ArrayFire), lo que nos ha permitido obtener finalmente más de tres órdenes de magnitud de mejora con respecto a la versión original.

Como líneas de trabajo futuras se plantean entre otras: i) optimización de las llamadas en CUDA; y ii) implementación heterogénea que haga uso tanto de la CPU como de la GPU, simultáneamente.

Agradecimientos

El presente trabajo ha sido financiado mediante el proyecto TIN2016-80920-R del Ministerio de Economía, Industria y Competitividad y por la Universidad de Málaga (Campus de Excelencia Internacional Andalucía Tech). También agradecemos al BSC que haya puesto a nuestra disposición el servidor de altas prestaciones MinoTauro.

Referencias

[1] Alan V. Oppenheim, "Signals and systems," *Prentice-hall Signal Processing Series, Prentice Hall*, 1996.
[2] "FFTW," `http://www.fftw.org/`, Accessed: 2019-05-23.
[3] Gabriele Jost Barbara Chapman and Ruud van der Pas, "Using openmp – portable shared memory parallel programming," *The MIT Press*, october 2007.
[4] "ArrayFire," `https://arrayfire.com/`, Accesses: 2019-05-23.
[5] A.F. Bell, S. Hernandez, H.E. Gaunt, P. Mothes, M. Ruiz, D. Sierra, and S. Aguaiza, "The rise and fall of periodic 'drumbeat' seismicity at tungurahua volcano, ecuador," *Earth and Planetary Sci. Lett.*, vol. 475, pp. 58–70, 2017.
[6] A.F. Bell, M. Naylor, S. Hernandez, I.G. Main, H.E. Gaunt, P. Mothes, and M. Ruiz, "Volcanic eruption forecasts from accelerating rates of drumbeat long-period earthquakes," *Geophysical Research Letters*, 2018.
[7] "MinoTauro User's Guide," `https://www.bsc.es/support/MinoTauro-ug.pdf`, Accessed: 2019-05-23.
[8] M. Beyreuther, R. Barsch, L. Krischer, T. Megies, Y. Behr, and J. Wassermann, "Volcanic eruption forecasts from accelerating rates of drumbeat long-period earthquakes," *Geophys*, vol. 81, no. 3, pp. 530–533, 2018.
[9] "FFTW 3.3.8 Documentation," `http://www.fftw.org/fftw3.pdf`, Accessed: 2019-05-23.
[10] "Ctypes Documentation," `https://docs.python.org/3/library/ctypes.html`, Accesses: 2019-05-25.
[11] "ArrayFire Documentation," `http://arrayfire.org/docs/index.htm`, Accesses: 2019-05-23.
[12] Avery Li-Chun Wang, "An industrial-strength audio search algorithm," *Shazam Entertainment, Ltd.*, 2003.
[13] J. Ruiz-Ferrer, A. Vilches, O. Torreno, and D. Cuesta, "Khiva: Accelerated time-series anlaytics on GPUs and CPU multicores," 2018, `https://github.com/shapelets/khiva`.
[14] Kurt W. Smith, "Cython, a guide for python programmers," *O'Reilly*, 2015.

Medición de overheads para el uso eficiente de recursos en centros de computación de alto rendimiento

Javier Corral-García[1], José-Luis González-Sánchez[1] y Miguel-Ángel Pérez-Toledano[2]

Resumen— **El desarrollo de códigos para su ejecución en infraestructuras de computación de alto rendimiento (HPC, High-Performance Computing) puede resultar una ardua tarea para científicos que no son expertos en programación paralela, pero que necesitan estas infraestructuras para el desarrollo de sus investigaciones. A menudo intentan ejecutar sus códigos utilizando tantos núcleos como sea posible, sin considerar el overhead asociado, creyendo, en muchos casos erróneamente, que de ese modo obtendrán antes sus resultados, con el consecuente e innecesario gasto que esto puede suponer en términos de tiempo, energía y recursos. Éste es un problema común en los centros de HPC, donde las decisiones de planificación eficiente y el ahorro energético se convierten en desafíos clave que afrontar diariamente. Para ayudar a resolverlos, se presenta un método de medición de overhead, centrado en la eficiencia, para analizar los efectos producidos, tanto por el incremento del número de cores de ejecución, como por la planificación paralela escogida en cada caso.**

Palabras clave— **HPC; High-Performance Computing; Overhead; OpenMP; Paralelización automática.**

I. Introducción

Científicos e investigadores de todo el mundo se enfrentan diariamente a desafíos críticos que requieren el uso de la computación de alto rendimiento para su resolución. Sin embargo, la notable complejidad de la programación paralela condiciona la eficiencia de los códigos desarrollados por estos expertos, que a menudo presentan notables dificultades para explotar adecuadamente los beneficios que ofrecen este tipo de infraestructuras. Esto también supone importantes problemas para los administradores de centros de HPC, donde el uso adecuado de los recursos y el consumo eficiente de energía suponen desafíos claves.

Para responder a estos problemas, algunos centros utilizan sus propias herramientas y métodos. En el caso de CénitS [1] (el Centro Extremeño de iNvestigación, Innovación Tecnológica y Supercomputación) se decidió desarrollar un transcompilador [2], enfocado a la formación de nuevos usuarios en el ámbito de la programación paralela, que persigue equilibrar el uso eficiente de los recursos en este tipo de centros, con la necesidad de sus usuarios de obtener resultados en tiempos adecuados, mediante la paralelización automática de códigos secuenciales en lenguaje C [3] transformados en códigos paralelos OpenMp (Open Multi-Processing) [4].

Dado que una implementación paralela adecuada precisa conocer de antemano cómo y dónde se emplea el tiempo de ejecución requerido, era necesario que el transcompilador dispusiera de información previa sobre los tiempos de overhead generados durante la ejecución de determinadas operaciones paralelas, en función del esquema de planificación empleado que controla la asignación de tareas a cada hilo de ejecución. Sin embargo, los enfoques sobre medición de overheads propuestos en la literatura no tienen en cuenta la eficiencia y, por lo tanto, no están orientados a analizar los overheads producidos por determinadas operaciones, ni a analizar los efectos negativos generados por el incremento del número de cores utilizados.

El resto del documento está organizado como se describe a continuación: la siguiente sección presenta el alcance y las principales motivaciones y objetivos del enfoque. La sección III discute el trabajo relacionado. La sección IV contiene un breve resumen sobre aspectos esenciales de OpenMP, mientras que la Sección V ofrece una descripción detallada del método propuesto. Los resultados experimentales se presentan y analizan en la Sección VI. Finalmente, la Sección VII concluye y resume las principales aportaciones del documento.

II. Alcance y objetivos

Elaborar trabajos que sean ejecutados en un centro de supercomputación puede resultar complicado, especialmente para científicos que no presentan experiencia previa en computación paralela. El transcompilador desarrollado en CénitS para la paralelización automática de códigos secuenciales (mencionado anteriormente) persigue que programadores sin experiencia en la programación de códigos paralelos puedan emplear recursos de HPC de forma eficiente. Sin embargo, aunque su utilización mejora notablemente el rendimiento de los códigos paralelizados, aún se requieren mejoras en el módulo de optimización que determina, tanto la planificación de ejecución, como el número adecuado de cores que pueden ser utilizados para hacer un uso responsable de los recursos disponibles.

[1]COMPUTAEX - CénitS (Centro Extremeño de iNvestigación, Innovación Tecnológica y Supercomputación), Cáceres, España, e-mail: {javier.corral}, {joseluis.gonzalez}@cenits.es

[2]Dpto. de Ingeniería de Sistemas Informáticos y Telemáticos. Universidad de Extremadura. Cáceres, España, e-mail: toledano@unex.es

Cuando un programa invoca un bucle paralelo incurre en cierto overhead consistente en el tiempo requerido para coordinar sus tareas, en el cual influyen varios factores, como la sincronización o las comunicaciones de datos entre los distintos hilos de ejecución. Aunque no resulta sencillo prever el overhead asociado en cada situación, un bucle no debe ser paralelizado o sus tareas distribuidas entre demasiados hilos a menos que su ejecución paralela requiera menos tiempo que dicho overhead. De lo contrario, esto se traduce en un gasto innecesario de tiempo, energía y recursos. Sin embargo, es habitual que los usuarios intenten ejecutar sus códigos paralelos utilizando siempre tantos cores como sea posible.

Por ello, el objetivo principal del método de medición propuesto es poder estimar con precisión los costes de tiempo asociados a ciertas operaciones clave en codigos paralelos OpenMP, en función del número de cores utilizados y las estrategias de planificación escogidas, con el fin de mejorar su eficiencia. Dado que el transcompilador está centrado en la paralelización automática de bucles (ya que la mayoría de los programas, especialmente las aplicaciones científicas, emplean la mayor parte del tiempo en ellos), las mediciones presentadas también están orientadas a estas estructuras de control, aunque se prevé extender su funcionalidad a otras estructuras en futuras versiones.

III. Trabajos relacionados

El tiempo de ejecución es uno de los recursos principales para conocer el comportamiento de un programa paralelo. Las estrategias de análisis de rendimiento suelen analizar el tiempo empleado por cada línea de código o función, para encontrar posibles oportunidades de optimización. Sin embargo, las basadas en instrumentación de código provocan significativos overheads adicionales especialmente ante la existencia de funciones con reducidos tiempos de ejecución [5], mientras que las técnicas de muestreo son menos invasivas pero con resultados más imprecisos, al tratarse de aproximaciones estadísticas.

Una de las herramientas de *function profiling* más utilizadas es *gprof* [6], que emplea un híbrido entre instrumentación y muestreo. Sin embargo este método también sufre importantes problemas si el código contiene grandes funciones que contribuyen significativamente al incremento del tiempo de ejecución final. Además, cuando el propio compilador realiza el análisis del rendimiento, insertando el código de una función en el lugar donde es empleado, los resultados pueden ser malinterpretados.

También existen herramientas basadas en analizar líneas individuales del código, pero presentan problemas con la reorganización del mismo que realizan los compiladores. Además, puede resultar imposible conocer el tiempo de ejecución de líneas específicas, debido a la arquitectura en *pipeline* de los microprocesadores actuales.

Otra opción consistiría en la utilización de contadores de rendimiento hardware, sin embargo, su número es muy limitado en algunos procesadores, por lo que su utilización fue descartada.

Por otro lado, existen diversos benchmarks basados en OpenMP [7–15], pero no están diseñadas para calcular overheads sino para evaluar los mecanismos de planificación de tareas. Se puede encontrar una amplia revisión bibliográfica sobre estos benchmarks en [14]. Entre ellos, únicamente los microbenchmarks EPCC [16], ampliamente utilizados, pueden ayudar a los investigadores a estimar los overheads asociados al uso de diferentes construcciones de OpenMP [17]. Sin embargo, no se centran en estudiar los overheads de ciertas operaciones ni en analizar los efectos de incrementar el número de cores utilizados, sino en los efectos del tamaño de los paquetes de trabajo o fragmentos (*chunks*) de iteraciones distribuidos entre los distintos hilos. Aunque estos microbenchmarks fueron ampliados en [18], las mejoras se centraron en medir los overheads asociados a usos comunes de tareas OpenMP. Sphinx [10] es otro conjunto de microbenchmarks, que proporciona un entorno para la medición de overheads, pero que considera únicamente los costes de utilizar OpenMP junto con MPI [19].

En resumen, se han propuesto diversos enfoques para la medición de overheads en OpenMP, pero ninguno lo estima con precisión en función del número de cores utilizados y las estrategias de ejecución paralela empleadas en cada caso. Por ello, el trabajo presentado en este artículo pretende cubrir dicha ausencia.

IV. OpenMP

La presente propuesta se centra en la medición de overheads en códigos OpenMP, por ser esta última la API de programación utilizada en el transcompilador para ofrecer la paralelización automática, debido a su facilidad de aprendizaje para usuarios no expertos respecto a MPI (Message Passing Interface), al no requerir que las estructuras de datos del programa estén particionadas explícitamente [20].

Para la medición de overheads, se han tenido en cuenta los tres tipos de planificación que ofrece OpenMP para realizar la asignación de tareas a hilos o procesos, cuya elección puede tener un importante impacto en el rendimiento [21]: estática, dinámica, y guiada.

- *Static*: las iteraciones se dividen en fragmentos (*chunks*) del tamaño especificado o lo más iguales posibles en tamaño, que son asignados de forma estática a los hilos mediante *round-robin*, siguiendo el orden de los hilos. Como consecuencia, el último fragmento puede presentar menos iteraciones.
- *Dynamic*: útil para manejar cargas de trabajo poco equilibradas e impredecibles. Las iteraciones se asignan desde la cola de trabajo a los hilos cuando éstos solicitan otro fragmento, hasta que no quede ninguno.

- *Guided*: similar a la planificación dinámica, el tamaño del fragmento disminuye con el tiempo, con objeto de reducir la sobrecarga procesando los fragmentos más grandes al principio. Así, según se va acercando el final de la ejecución, el sistema va utilizando fragmentos más reducidos para completar huecos en la planificación.

Existe también una cuarta opción, *Runtime* [21], aunque ha sido descartada por no tratarse realmente de un esquema de planificación, sino de una forma de elegir uno de los tres esquemas anteriores en tiempo de ejecución.

La planificación más adecuada en cada caso depende de varios factores, entre los que se incluyen el código existente dentro de cada bucle, el tamaño del problema, el número de hilos (y cores) empleados y la propia carga del sistema. Por ello, es preciso lograr un equilibrio entre el uso de la memoria y el balanceo de la carga, mediante medidas de rendimiento, para descubrir qué método produce los mejores resultados. Una tarea complicada para científicos e investigadores que no son expertos en programación paralela [2].

V. Rendimiento paralelo y overhead

Es posible medir el rendimiento paralelo de una aplicación comparando el tiempo de su ejecución secuencial con el tiempo de su ejecución paralela. El *speedup*, $S_p(n)$ es la proporción existente entre ambos, definiéndose como $S_p = \frac{T}{T_p}$, donde T es el tiempo de ejecución secuencial, T_p es el tiempo paralelo y p es el número de procesadores utilizados.

La ley de Amdahl establece un límite teórico sobre el beneficio obtenido al aumentar el número de cores de procesamiento. Considerando que, siendo s la parte secuencial y p la parte paralela de un código, un programa tardaría una unidad de tiempo en ejecutarse $p + s = 1$, siendo su $Speedup = \frac{1}{S+\frac{1-S}{n}}$, donde S sería el tiempo empleado por la parte secuencial y n el número de núcleos de procesamiento, asumiendo una escalabilidad perfecta. La siguiente fórmula considera además el overhead adicional introducido como consecuencia de la paralelización del código:

$$Speedup = \frac{1}{S + \frac{1-S}{n} + kn + \lambda}$$

siendo kn el overhead de comunicación y λ el overhead n -independiente, pudiendo deducirse fácilmente que estos valores influyen negativamente en el rendimiento. De este modo, cada comienzo o finalización de la ejecución de una región paralela o un bucle for tiene un coste de overhead asociado, debido a que los hilos deben generarse o despertarse del estado inactivo en que se encontraban, se requiere determinar el tamaño de los paquetes de trabajo para cada hilo (en el caso de las planificaciones dinámica o guiada), se necesita ir asignando nuevas tareas a aquellos hilos que vayan quedando libres durante la ejecución, y además, las barreras predeterminadas al final de cada región paralela deben volver a sincronizar todos los hilos [5].

A. Método de medición de overhead

El método de medición desarrollado está basado en la suite de microbenchmarks EPCC para OpenMP [16, 17, 22] que, entre otras características, analiza los efectos de modificar el tamaño de los paquetes de trabajo (fragmentos de iteraciones) distribuidos entre los hilos. La presente propuesta permite analizar de forma precisa los efectos de aumentar el número de hilos (o cores) empleados, en función de la planificación escogida (estática, dinámica o guiada). Para ello se compara, el tiempo empleado por una sección de código ejecutada secuencialmente, con el tiempo necesario para su ejecución paralela, mientras se incrementa el número de cores (desde uno hasta el máximo ofrecido por la infraestructura), utilizando los tres tipos de planificación OpenMP descritos anteriormente. En ausencia de overhead, ambas ejecuciones (secuencial y paralela) deberían tardar el mismo tiempo. Así, con el objetivo de estudiar el overhead de la distribución paralela de tareas mediante bucles for, el método mide en primer lugar el tiempo necesario para ejecutar el siguiente código paralelo con una planificación (*schedule*) específica y un número concreto de hilos (*threads*):

```
#pragma omp parallel private(i)
for (i=0; i<repetitions; i++){
  #pragma omp for schedule(...)
  for (j=0; j<(iterations * threads); j++){
    Processing(time);
```

para posteriormente restar el tiempo necesario para ejecutar el siguiente código secuencial en un solo hilo:

```
for (i=0; i<repetitions; i++){
  for (j=0; j<(iterations); j++){
    Processing(time);
```

y dividir el resultado por el número total de *repeticiones*, cuyo valor es calculado previamente para que una ejecución completa alcance exactamente el tiempo específico elegido por el usuario (en adelante T_{exe}). Por simplicidad, la constante *iterations* tiene un valor de 128, determinado previamente por su adecuación para obtener valores medición precisos, al haber demostrado estabilidad en la obtención de resultados tanto en pequeñas ejecuciones como en otras más considerables, sin apenas variaciones entre ellas. Además, la función *Processing* no consume caché o ancho de banda de memoria, al consistir únicamente en el incremento de una variable tantas veces como sean necesarias para que la ejecución alcance el tiempo de ejecución indicado por el usuario. Cuando se requiere estudiar el overhead de determinadas operaciones paralelas dentro de bucles for, esta función es reemplazada por dichas operaciones, como se describe más adelante en los apartados VI-B y VI-C. Cada test debe ser ejecutado en repetidas ocasiones, con el fin de obtener datos estadísticos precisos con desviaciones despreciables.

Como resultado, el método muestra, para cada test, los tiempos de ejecución medio, mínimo y máximo, la desviación estándar, el número de valores atípicos obtenidos y un intervalo de confianza del 95 %, junto con el overhead añadido respecto a la ejecución secuencial.

VI. Resultados experimentales

Un objetivo prioritario a la hora de desarrollar, tanto el método de medición como los propios experimentos, ha sido alcanzar un alto nivel de precisión en las mediciones. Esta precisión está determinada no solo por el número de mediciones realizadas (en adelante *nmeas*), sino también por la duración total de cada test (definida previamente como T_{exe}). De este modo, cada test es repetido *nmeas* veces, mientras que las mediciones son realizadas cuando un test comienza o finaliza su ejecución, tal y como muestra el siguiente código:

```
for (i=0; i<nmeas; i++){
   start = getclock();
   test();
   end = getclock();
```

Para estudiar la influencia de ambas variables (*nmeas* y T_{exe}) en la precisión de los resultados, se realizaron pruebas con distintas combinaciones, considerando la utilización de 1 a 20 cores de ejecución, utilizando los tres tipos de planificación (estática, dinámica y guiada), con funciones (*Processing*) de 15, 30, 45, 50, 75 y 90 μs de duración, cuatro posible valores de repeticiones (*nmeas*) de cada test (10^3, 10^4, 10^5 o 10^6 mediciones) y tres duraciones totales de T_{exe} (10 ms, 100 ms o 1,000 milisegundos). Los resultados de este estudio previo mostraron que la utilización de valores altos en T_{exe} tenían una influencia positiva en la precisión mayor que cuando se incrementaba *nmeas*. Así, los resultados con menor desviación típica y menor número de valores atípicos fueron obtenidos con un tiempo de 10^6 ms por ejecución, y un total de 1.000 milisegundos $n \times 10^3$ mediciones. Con otros valores o combinaciones, el nivel de incertidumbre aumentaba considerablemente.

Los experimentos descritos a continuación fueron realizados en un servidor Fujitsu Primergy CX2550 con 2 procesadores Intel Xeon E5-2660v3, con 10 cores por procesador (20 cores en total), a 2,6GHz con 25 MB de caché, 80GB de memoria RAM y dos discos SSD de 128GB. Se eligió esta infraestructura por resultar representativa, siendo ampliamente utilizada en centros de supercomputación y una de las más requeridas por los investigadores a la hora de realizar sus ejecuciones en CénitS. Sin embargo, el método puede ser aplicado en otro tipo de infraestructuras y arquitecturas computacionales, estando previsto su estudio en futuros trabajos, con el objetivo de que los resultados puedan ser extrapolados a otros centros de HPC para mejorar el uso eficiente de sus recursos.

Se realizaron tres tipos de experimentos para obtener datos sobre los siguientes aspectos:

a) Overhead en bucles *for*: para calcular adecuadamente el overhead asociado a la utilización de bucles paralelos en función de la cantidad de núcleos de ejecución y la planificación paralela empleada.
b) Overheads asociados a la ejecución de operaciones computacionales representativas dentro de bucles paralelos: para obtener información sobre cómo influye cada tipo de operación en el overhead asociado.
c) Overheads en algoritmos paralelos conocidos: para verificar las conclusiones obtenidas con las mediciones anteriores mediante programas completamente funcionales.

A. Overhead en bucles for

La Tabla I muestra el overhead (en microsegundos por iteración) asociado a los bucles *for* medidos para un número de hilos (un hilo por core) de 1 a 20 (mostrando en la tabla únicamente los pares, por razones de legibilidad), con los tres tipos de planificación (estática, dinámica y guiada) y diferentes tiempos de procesamiento.

Con el objetivo de demostrar que el tiempo de ejecución de la función *Processing* afecta al overhead de forma constante (en función del tiempo de procesamiento), se presenta la Tabla II que muestra los resultados de dividir los overheads de la Tabla I entre los tiempos de procesamiento correspondientes (15, 30, 45, 60, 75 y 90). Los resultados parciales han sido redondeados a dos decimales para facilitar su comprensión.

Como se puede observar, los incrementos en los tiempos de procesamiento penalizan los resultados e influyen en el overhead de forma constante, dependiendo además del número de hilos de ejecución, de forma que el factor de overhead permanece constante siempre que se utilice el mismo número de hilos.

Así, por ejemplo, el overhead asociado a la ejecución de un bucle *for* con 10 hilos con un tiempo de ejecución de 60 μs, podría calcularse como el resultado de multiplicar el factor de overhead para 12 hilos $(0, 146)$ por el tiempo de procesamiento (60 μs), dando un overhead de 8,76 μs (con una desviación de 0,25 μs respecto al valor real).

TABLA I: Overhead en bucles *for* (μs por iteración) según planificación, tiempo de procesamiento y número de hilos.

Processing time (μs)			Threads									
			2	**4**	**6**	**7**	**10**	**12**	**14**	**16**	**18**	**20**
	15	static	0.02	0.20	1.21	1.71	2.24	2.24	2.25	2.25	2.25	2.27
		dynamic	0.09	0.20	1.24	1.82	2.33	2.33	2.33	2.33	2.33	2.35
		guided	0.03	0.20	1.19	1.78	2.27	2.28	2.28	2.29	2.29	2.31
	30	static	0.04	0.72	2.36	3.48	4.37	4.37	4.37	4.37	4.37	4.41
		dynamic	0.11	0.53	2.48	3.49	4.44	4.45	4.45	4.45	4.46	4.47
		guided	0.04	0.34	2.50	3.47	4.40	4.41	4.41	4.41	4.42	4.45
	45	static	0.05	1.05	3.42	5.07	6.39	6.39	6.40	6.40	6.41	6.45
		dynamic	0.12	0.45	3.42	5.05	6.46	6.46	6.46	6.47	6.47	6.50
		guided	0.05	0.49	4.09	5.06	6.42	6.42	6.43	6.44	6.45	6.46
	60	static	0.08	0.68	5.19	6.80	8.51	8.51	8.51	8.51	8.51	8.57
		dynamic	0.13	0.60	4.51	6.73	8.57	8.58	8.59	8.58	8.60	8.61
		guided	0.07	0.63	4.56	6.76	8.54	8.54	8.55	8.55	8.55	8.59
	75	static	0.08	1.13	6.13	9.03	11.32	11.33	11.33	11.33	11.38	11.43
		dynamic	0.18	0.81	5.99	9.94	11.40	11.39	11.40	11.40	11.40	11.46
		guided	0.11	1.74	6.02	8.99	11.35	11.35	11.36	11.37	11.37	11.43
	90	static	0.08	1.03	6.68	9.89	12.47	12.47	12.47	12.46	12.47	12.56
		dynamic	0.18	1.30	6.61	9.85	12.52	12.54	12.54	12.53	12.54	12.58
		guided	0.12	1.80	6.62	9.86	12.49	12.49	12.52	12.52	12.52	12.55

TABLA II: Factor de overhead en bucles for en función de planificación, tiempo de procesamiento y número de hilos.

Processing time (μs)			Threads									
			2	4	6	8	10	12	14	16	18	20
	15	static	0.00	0.01	0.08	0.12	0.15	0.15	0.15	0.15	0.15	0.15
		dynamic	0.01	0.01	0.08	0.12	0.16	0.16	0.16	0.16	0.16	0.16
		guided	0.00	0.01	0.08	0.12	0.15	0.15	0.15	0.15	0.15	0.15
	30	static	0.00	0.02	0.08	0.12	0.15	0.15	0.15	0.15	0.15	0.15
		dynamic	0.00	0.02	0.08	0.12	0.15	0.15	0.15	0.15	0.15	0.15
		guided	0.00	0.01	0.08	0.12	0.15	0.15	0.15	0.15	0.15	0.15
	45	static	0.00	0.02	0.08	0.11	0.14	0.14	0.14	0.14	0.14	0.14
		dynamic	0.00	0.01	0.08	0.11	0.14	0.14	0.14	0.14	0.14	0.14
		guided	0.00	0.01	0.09	0.11	0.14	0.14	0.14	0.14	0.14	0.14
	60	static	0.00	0.01	0.09	0.11	0.14	0.14	0.14	0.14	0.14	0.14
		dynamic	0.00	0.01	0.08	0.11	0.14	0.14	0.14	0.14	0.14	0.14
		guided	0.00	0.01	0.08	0.11	0.14	0.14	0.14	0.14	0.14	0.14
	75	static	0.00	0.02	0.08	0.12	0.15	0.15	0.15	0.15	0.15	0.15
		dynamic	0.00	0.01	0.08	0.12	0.15	0.15	0.15	0.15	0.15	0.15
		guided	0.00	0.02	0.08	0.12	0.15	0.15	0.15	0.15	0.15	0.15
	90	static	0.00	0.01	0.07	0.11	0.14	0.14	0.14	0.14	0.14	0.14
		dynamic	0.00	0.01	0.07	0.11	0.14	0.14	0.14	0.14	0.14	0.14
		guided	0.00	0.02	0.07	0.11	0.14	0.14	0.14	0.14	0.14	0.14
	Media:		0.002	0.015	0.079	0.115	0.146	0.146	0.146	0.146	0.146	0.147

Estos overheads están asociados a una única iteración, pero pueden llegar a suponer tiempos significativos cuando el número de iteraciones crece, como en el caso de programas que requieren ser ejecutados durante días o incluso semanas.

La Figura 1 muestra los overheads asociados a un tiempo de procesamiento de 15 μs y 128 iteraciones del bucle para cada planificación: estática (representada en azul), dinámica (rojo) y guiado (verde). Con la utilización de pocos cores (entre 2 y 4) el overhead tiene un efecto moderado, pero a medida que crece su número el overhead aumenta también notablemente, permaneciendo constante a partir de los 10 hilos, no existiendo mucha diferencia, en términos del overhead asociado, entre compartir el trabajo con 10 o 20 hilos. Además, aunque no existe mucha diferencia entre cada esquema, la planificación estática implica overheads ligeramente inferiores. Como ejemplo, empleando 10 o más hilos, este caso particular presenta un overhead de 287 μs, por lo que un bucle que ejecutado de forma secuencial no exceda ese tiempo no debería ser paralelizado.

En relación a los tres tipos de planificaciones, en la dinámica o guiada, la asignación varía en tiempo de ejecución, porque las iteraciones no se asignan únicamente al inicio del bucle, sino a lo largo de toda su ejecución, siempre que un hilo requiera más iteraciones tras finalizar su tarea parcial. El handicap reside en la coordinación que debe realizar el sistema para garantizar que cada iteración sea ejecutada exactamente una vez, con el consiguiente coste de sincronización. En la planificación estática,

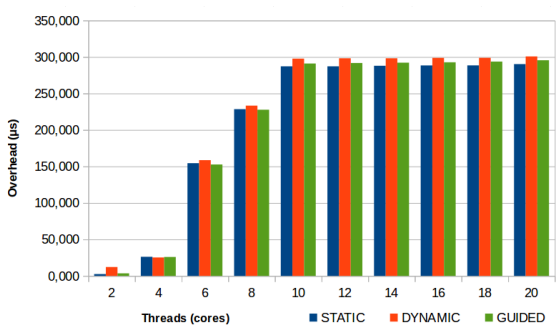


Fig. 1: Overhead asociado a un tiempo de procesamiento de 15 μs y 128 iteraciones

cada hilo ejecuta las iteraciones que tiene asignadas desde el principio, sin que se produzca ningún cambio adicional en tiempo de ejecución. Por ello, esta planificación suele generar menos overhead, al evitar este coste de sincronización, aunque a cambio no puede compensar los desequilibrios que se produzcan en las cargas de trabajo asignadas.

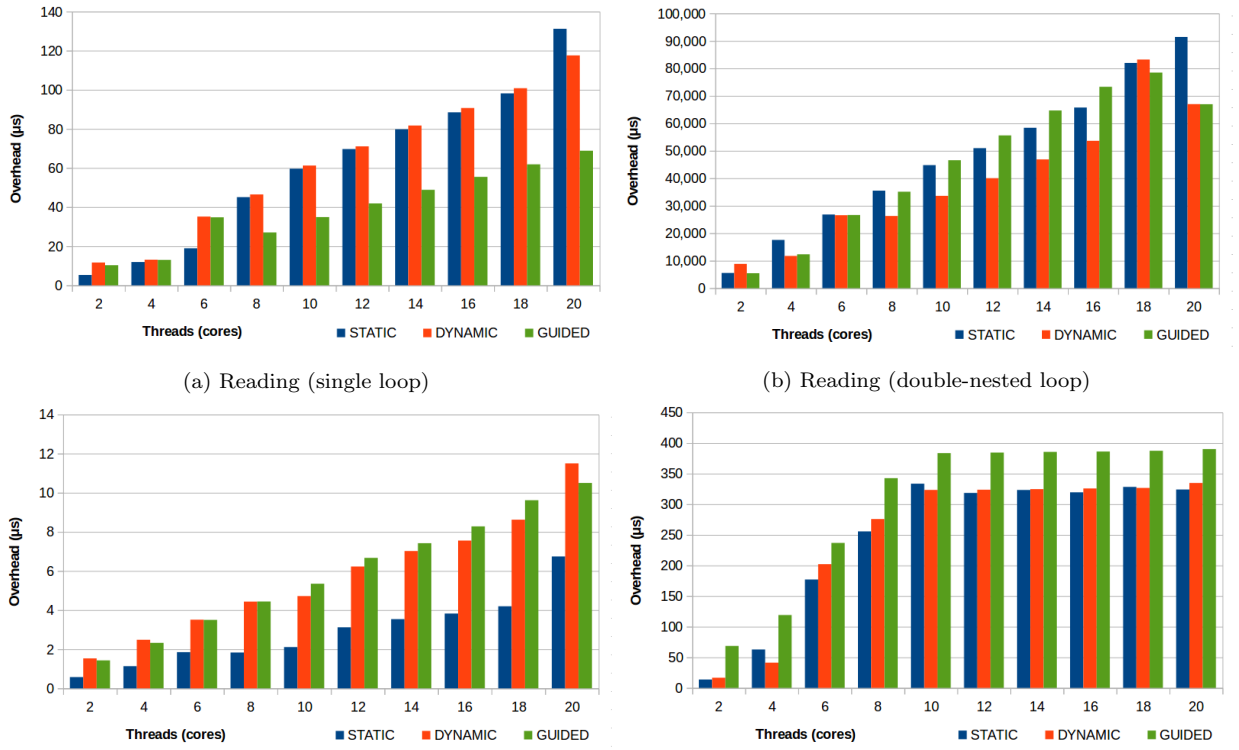
B. Overhead en operaciones representativas

A continuación se muestran los resultados de calcular el overhead en las siguientes operaciones paralelas, divididas en dos tipos de bucles *for* e implicando matrices.

```
/* (1) single loop, assignment */
for (i = 0; i < n; i++)
        a[i] = value;

/* (2) single loop, reading */
for (i = 0; i < n; i++)
        aux = a[i];

/* (3) single loop, addition */
for (i = 0; i < n; i++)
        a[i] = b[i] + c[i];

/* (4) single loop, reduction */
for (i = 0; i < n; i++)
        result = result + a[i];

/* (5) double-nested loop, assignment */
for (i = 0; i < n; i++)
        for (k = 0; k < n; k++)
                a[i][k] = value;

/* (6) double-nested loop, reading */
for (i = 0; i < n; i++)
        for (k = 0; k < n; k++)
                aux = a[i][k];

/* (7) double-nested loop, addition */
for (i = 0; i < n; i++)
        for (k = 0; k < n; k++)
                a[i][k] = b[i][k]+c[i][k];

/* (8) double-nested loop, reduction */
for (i = 0; i < n; i++)
        for (k = 0; k < n; k++)
                result = + a[i][k];
```

En todos los casos, $n = 1.000$ y una directiva *pragma omp for* controla el bucle que distribuye la misma carga de trabajo entre todos los hilos. De este modo, es posible comparar el tiempo secuencial y el paralelo, para calcular el overhead producido. Las matrices y variables han sido generadas previamente empleando *Uniform* C [23], para la generación de secuencias de números pseudoaleatorios de precisión real doble distribuidos uniformemente.

La Figura 2 muestra los resultados más representativos obtenidos. Como cabría esperar, los bucles doblemente anidados producen overheads mucho mayores que los simples. Particularmente destacables son los datos de las operaciones de lectura, con overheads ampliamente superiores que en el resto de los casos (5.454,93 microsegundos con solo dos hilos). Por contra, la operación de reducción con planificación estática y un único bucle es la que produce menos overhead, con una diferencia además de solo 6,17 microsegundos entre el uso de 2 o 20 hilos. En este último caso, existe también una importante variación entre los overheads generados por las distintas estrategias, con un incremento



(a) Reading (single loop)

(b) Reading (double-nested loop)

(c) Reduction (single loop)

(d) Reduction (double-nested loop)

Fig. 2: Overhead en operaciones paralelas representativas.

medio entre el 53 y el 59 % si no se utiliza la planificación estática.

Es precisamente dicha estrategia la que obtiene mejores resultados en el 50 % de los casos y en 3 de los 4 experimentos con bucles simples. Sin embargo, no sería apropiado aplicar siempre esta planificación, puesto que también presenta overheads muy elevados en las operaciones de lectura y en las de asignación y suma con bucles dobles.

En resumen, se obtuvieron las siguientes conclusiones: las operaciones de lectura generan los overheads más elevados (especialmente en bucles dobles: hasta 200 veces más que las operaciones de reducción, y aproximadamente 100 veces más que en el resto de los casos), mientras que la operación de reducción es la más económica en términos de overhead. Con respecto a los esquemas de planificación disponibles y con una única operación dentro del bucle:

- Es muy recomendable emplear la planificación estática en casos de reducción con un bucle simple, y debe usarse en operaciones de asignación o suma con un solo bucle y en reducciones con bucles dobles. Sin embargo, debe evitarse en las asignaciones con bucles dobles y en lecturas (bucles simples y dobles).
- La planificación dinámica es muy recomendable en casos de lectura y suma con bucle doble, y debe evitarse en lecturas y sumas con bucles simples.
- La planificación guiada logra los mejores resultados en la asignación con bucle doble y lectura con un solo bucle, y debe evitarse en la reducción (bucle simple y doble).

En el futuro se realizarán estudios más detallados sobre los efectos de combinar diferentes operaciones dentro de cada bucle.

C. Influencia del overhead en algoritmos conocidos

Los siguientes experimentos se realizaron con objeto de demostrar la influencia del overhead en los tiempos de ejecución. Para ello, se compararon los incrementos producidos en el overhead al aumentar el número de hilos, con las variaciones en los tiempos de ejecución finales. Se utilizaron problemas conocidos cuyas paralelizaciones representan importantes mejoras de tiempo respecto a sus versiones secuenciales. De nuevo, el trabajo es incrementado en base al número de hilos empleados, con el objetivo de compartir siempre dicho trabajo de forma equitativa.

(1) Matrix multiplication: calcula un producto de matrices densas ($C = A \times B$), de matrices cuadradas de orden 100:

```
#pragma omp for schedule() private(i,k,z)
for (i=0; i<n; i++) {
   for (k=0; k<n; k++) {
      a[k][i] = 0.0;
      for (z=0; z<n; z++) {
         a[k][i] = a[k][i] + b[k][z] * c[z][i];
      }
   }
}
```

(2) Buscador de números primos: muestra los números primos entre 1 y 10^4, con el bucle principal paralelizado de la siguiente forma:

```
#pragma omp parallel for schedule() private(i,end,prime)
   shared(n) reduction(+:j) reduction(+:total)
for (i=1; i<=n; i+=2){
    end = (int) sqrt((float)i) + 1;
    prime = 1;
    j = 3;
    while (prime && (j<=end)){
       if (i % j == 0) prime = 0;
       j+= 2;
    }
}
```

(3) Estimación de la integral $\int_a^b f(x) = 50/(\pi \cdot (2,5 \cdot 10^2 \cdot x^2 + 1))\,\mathrm{dx}$ mediante un algoritmo de promedio de 10^4 iteraciones, con un margen de error de $7,9 \cdot 10^{-9}$.

```
#pragma omp for schedule() private (h,i,x) \
   reduction (+ : total)
for (h=0; h < itersperthr * threads; h++){
   for (i=0.0; i<n; i++) {
      x = ((double)(n-i-1)*a+(double)(i)*b)/(double)(n-1);
      total = total + (50.0/(pi*(2500.0*x*x + 1.0)));
   }
}
```

Las Figuras 3 a 5 representan los resultados de estos experimentos. Cada gráfico muestra en el eje horizontal el número de hilos (uno por core) utilizado, mientras que los tiempos (en microsegundos) son mostrados en el eje vertical. Se emplearon los tres esquemas de planificación: estático (representado por cuadrados azules) dinámico (círculos rojo) y guiado (triángulo verde). En los tres experimentos, las variaciones de overhead tienen un impacto similar en las variaciones de los tiempos de ejecución. Es decir, los tiempos de ejecución aumentan proporcionalmente a los tiempos de overhead estimados, como se esperaba. Las mayores variaciones se producen a medida que el número de hilos aumenta de 1 a 10, mientras que de 10 a 20 hilos las variaciones del overhead son menos significativas.

VII. Conclusiones

Se ha presentado un método de medición de overhead para estimar con precisión su coste en operaciones paralelas representativas, en base al número de cores empleados y la estrategia de planificación escogida. Se ha demostrado su efectividad calculando overheads asociados a bucles *for* paralelos y a operaciones paralelas representativas, estimando su influencia en los tiempos de ejecución. De igual modo, también se describen recomendaciones generales para escoger, de antemano y en función de la operación paralela a ejecutar, la planificación estratégica más ventajosa según su overhead asociado. La efectividad del método de medición se ha demostrado empleando algoritmos conocidos (cuyas paralelizaciones representan importantes mejoras de tiempo respecto a sus ejecuciones secuenciales), comprobando que los tiempos de ejecución aumentan proporcionalmente a los overheads estimados, tal y como se esperaba.

Agradecimientos

Estudio cofinanciado por el Fondo Europeo de Desarrollo Regional (FEDER) Programa Operativo 2014-2020 de Extremadura, con un porcentaje del 80 %, dentro del Eje Prioritario 1 “Potenciar la investigación, el desarrollo tecnológico y la innovación” y el Eje Prioritario 2 “Mejorar el uso y la calidad de las tecnologías de la información y de la comunicación y el acceso a las mismas”, Proyecto CultivData Ref. 2018.14.02.332A.444.00. Este estudio también desea agradecer la cofinanciación recibida en el Proyecto RTI2018-094591-B-I00 al Ministerio de Ciencia, Innovación y Universidades (MCIU), a la Agencia Estatal de Investigación (AEI) y al Fondo Europeo de Desarrollo Regional (FEDER).

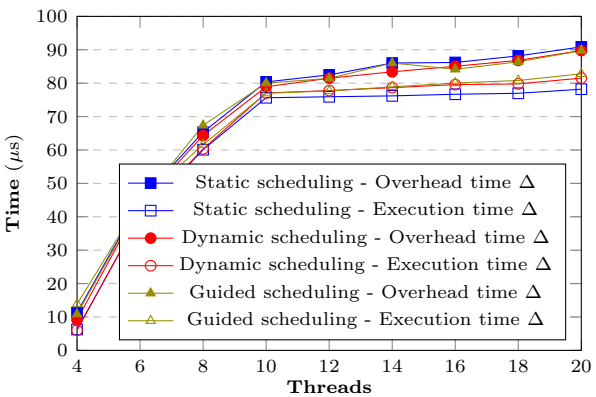


Fig. 4: Prime finder example. Variations of overhead with respect to variations of execution time.

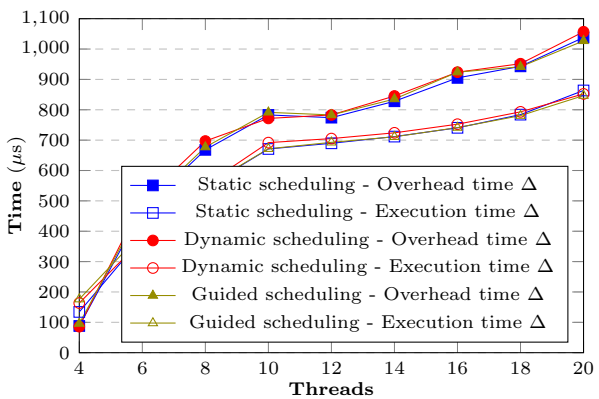


Fig. 3: Matrix multiplication example. Variations of overhead with respect to variations of execution time.

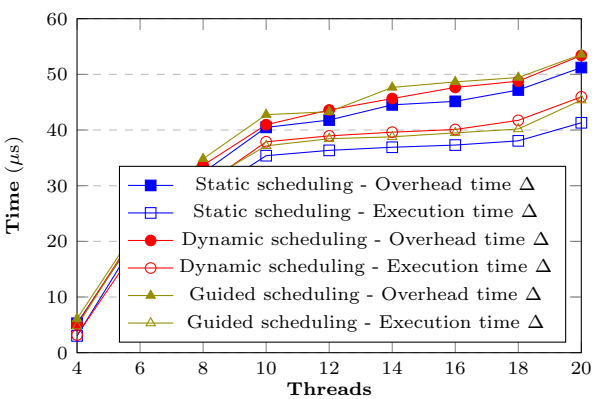


Fig. 5: Estimation of the integral example. Variations of overhead with respect to variations of execution time.

Referencias

[1] "CénitS: Extremadura Supercomputing, Technological Innovation and Research Center," http://www.cenits.es, 2019.

[2] J. Corral-García, J.L. González-Sánchez, and M.A. Pérez-Toledano, "Towards automatic parallelization of sequential programs and efficient use of resources in HPC centers," in *2016 International Conference on High Performance Computing Simulation (HPCS)*, July 2016, pp. 947–954.

[3] Dennis M Ritchie, Brian W Kernighan, and Michael E Lesk, *The C programming language*, Prentice Hall, 1988.

[4] "The OpenMP application program interface specification for parallel programming," http://www.openmp.org, July 2017.

[5] Georg Hager and Gerhard Wellein, *Introduction to high performance computing for scientists and engineers*, CRC Press, 2010.

[6] Susan L. Graham, Peter B. Kessler, and Marshall K. McKusick, "Gprof: A call graph execution profiler," *SIGPLAN Not.*, vol. 39, no. 4, pp. 49–57, Apr. 2004.

[7] "NAS parallel benchmarks," https://www.nas.nasa.gov/publications/npb.html, July 2017.

[8] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The parsec benchmark suite: Characterization and architectural implications," in *2008 International Conference on Parallel Architectures and Compilation Techniques (PACT)*, Oct 2008, pp. 72–81.

[9] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S. H. Lee, and K. Skadron, "Rodinia: A benchmark suite for heterogeneous computing," in *2009 IEEE International Symposium on Workload Characterization (IISWC)*, Oct 2009, pp. 44–54.

[10] "The sphinx parallel microbenchmark suite," https://computation.llnl.gov/casc/sphinx, July 2017.

[11] "Asc sequoia benchmark codes," https://asc.llnl.gov/sequoia/benchmarks, July 2017.

[12] Piotr R Luszczek, David H Bailey, Jack J Dongarra, Jeremy Kepner, Robert F Lucas, Rolf Rabenseifner, and Daisuke Takahashi, "The hpc challenge (hpcc) benchmark suite," in *Proceedings of the 2006 ACM/IEEE Conference on Supercomputing*, New York, NY, USA, 2006, SC '06, ACM.

[13] Vishal Aslot, Max Domeika, Rudolf Eigenmann, Greg Gaertner, Wesley B Jones, and Bodo Parady, "Specomp: A new benchmark suite for measuring parallel computer performance," in *International Workshop on OpenMP Applications and Tools*. Springer, 2001, pp. 1–10.

[14] A. Duran, X. Teruel, R. Ferrer, X. Martorell, and E. Ayguade, "Barcelona openmp tasks suite: A set of benchmarks targeting the exploitation of task parallelism in openmp," in *2009 International Conference on Parallel Processing*, Sept 2009, pp. 124–131.

[15] Xavier Teruel, Christopher Barton, Alejandro Duran, Xavier Martorell, Eduard Ayguadé, Priya Unnikrishnan, Guansong Zhang, and Raul Silvera, "Openmp tasking analysis for programmers," in *Proceedings of the 2009 Conference of the Center for Advanced Studies on Collaborative Research*. IBM Corp., 2009, pp. 32–42.

[16] J Mark Bull, Fiona Reid, and Nicola McDonnell, "A microbenchmark suite for openmp tasks," in *International Workshop on OpenMP*. Springer, 2012, pp. 271–274.

[17] J Mark Bull and Darragh O'Neill, "A microbenchmark suite for openmp 2.0," *ACM SIGARCH Computer Architecture News*, vol. 29, no. 5, pp. 41–48, 2001.

[18] James LaGrone, Ayodunni Aribuki, and Barbara Chapman, "A set of microbenchmarks for measuring openmp task overheads," in *International Conference on Parallel and Distributed Processing Techniques and Applications*, 2011, vol. 2, pp. 594–600.

[19] William Gropp, Ewing Lusk, and Anthony Skjellum, *Using MPI: portable parallel programming with the message-passing interface*, vol. 1, MIT press, 1999.

[20] R. Chandra, R. Menon, L. Dagum, D. Kohr, D. Maydan, and J. McDonald, *Parallel Programming in OpenMP*, Elsevier Science, 2000.

[21] Barbara Chapman, Gabriele Jost, and Ruud Van Der Pas, *Using OpenMP: portable shared memory parallel programming*, vol. 10, MIT press, 2008.

[22] J Mark Bull, "Measuring synchronisation and scheduling overheads in openmp," in *Proceedings of First European Workshop on OpenMP*, 1999, vol. 8, p. 49.

[23] "Uniform: A uniform random number generator (rng)," https://people.sc.fsu.edu/~jburkardt/c_src/uniform/uniform.html, July 2017.

[24] Shameem Akhter and Jason Roberts, *Multi-core programming*, vol. 33, Intel press Hillsboro, 2006.

Soporte OpenCL 2.0 para Intel TBB

Felipe Muñoz, José C. Romero, Alejandro Villegas, Ángeles Navarro, Andrés Rodríguez, Rafael Asenjo[1]

Resumen— **El objetivo de este artículo es mejorar la implementación del soporte para arquitecturas heterogéneas en la librería Intel Threading Building Blocks, TBB. Esta librería proporciona un nivel de abstracción sobre C++ implementando diferentes conjuntos de clases y paquetes. Por ejemplo, Flow Graph, es un conjunto de clases de TBB que facilita el desarrollo de aplicaciones de tipo flujo de datos en el que distintos nodos de un grafo procesan los datos según van estando disponibles. Uno de los nodos soportados en TBB es el `opencl_node`, el cual está diseñado para procesar los datos en GPUs compatibles con OpenCL 1.2. Nuestra contribución ha consistido en dotar a este nodo de las nuevas características que introduce la versión OpenCL 2.0, principalmente el soporte de *fine-grained Shared Virtual Memory* (SVM). Con SVM se facilita la colaboración entre la GPU y la CPU ya que se habilita el acceso a una misma región de memoria.**

Para valorar la eficacia de estos cambios se ha utilizado una versión paralela de ViVid, un algoritmo de detección de objetos que se ha adaptado para funcionar con el nuevo módulo de OpenCL. Comparando los resultados obtenidos al ejecutar ViVid con la versión 1.2 del nodo `opencl_node` y la 2.0 con SVM, encontramos que esta última puede mejorar los tiempos de ejecución heterogéneos al simplificar la gestión de coherencia caché.

Palabras clave— **Intel Threading Building Blocks, Flow Graph, OpenCL 2.0, arquitectura heterogénea, programación paralela.**

I. Introducción

La programación paralela representa el futuro de la computación y se ha convertido en el modelo de programación dominante en las arquitecturas actuales. Lleva años presente en la computación de altas prestaciones, sin embargo, con la aparición de los procesadores *multicore* en los ordenadores personales y, más aún, con el surgimiento de los *smartphones*, la programación paralela se ha extendido a todas las áreas de la computación.

Los procesadores *multicore* surgieron de la necesidad de solucionar el problema de las limitaciones físicas que se encontraron en el desarrollo de los procesadores *single-core* la pasada década. En aras de una mayor reducción del consumo energético, también se han popularizado recientemente arquitecturas heterogéneas *on-chip*, las cuales incorporan una GPU, o incluso una FPGA, junto con la CPU.

El problema que plantean estas nuevas arquitecturas heterogéneas estriba principalmente en la dificultad de programación de aplicaciones que exploten todos los recursos computacionales. Como desarrolladores preocupados por el rendimiento, somos conscientes de la importancia de paralelizar las aplicaciones para no dejar *cores* de CPU ociosos. Usando el mismo razonamiento, en muchos casos tiene también sentido usar los *cores* de GPU al tiempo que los de CPU, de forma que no queden recursos desaprovechados en el chip. Evidentemente, los costes de desarrollo de aplicaciones heterogéneas aumentan significativamente en comparación con los de aplicaciones secuenciales. Nuestro objetivo a largo plazo es simplificar la implementación de aplicaciones heterogéneas, haciendo uso de características de alto nivel proporcionadas por C++11 y la librería *Intel Threading Building Blocks* (Intel TBB) [1].

Intel TBB es una librería que facilita el aprovechamiento de los diferentes dispositivos de nuestra arquitectura paralela y heterogénea, sin necesidad de una elevada experiencia en programación basada en *threads*. Para ello proporciona plantillas paralelas para la gran mayoría de patrones típicos de paralelismo como son bucles (`parallel_for` y `parallel_do`), reducciones (`parallel_reduce`), *pipelines* y flujos de datos (Flow Graph).

Flow Graph incluye un conjunto de clases que nos permiten crear grafos que sean escalables y totalmente configurables.

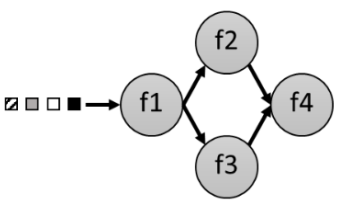


Fig. 1. Un posible grafo

Como vemos en la figura 1, usando el paralelismo con los grafos comunes, las unidades de computación pueden ser representadas por los nodos (f1, f2, f3 y f4) y los canales de comunicación entre estos, por los arcos que conectan los nodos. Una vez creado el grafo, cuando alguno de estos nodos recibe un mensaje, una tarea se genera para ejecutar la función implementada en el nodo el cual procesará el mensaje entrante. Una vez inicializado el grafo, los mensajes van fluyendo por el grafo. Estos mensajes pueden ser de cualquier naturaleza: matrices, imágenes, etc.

Para aprovechar todos los recursos presentes en nuestra máquina debemos ser capaces de mapear parte de la carga computacional de nuestra aplicación en cada uno de ellos. Uno de los recursos computacionales que queremos explotar es la GPU. A tal efecto, en este trabajo nos apoyaremos en el modelo de programación OpenCL (*Open Computing Language*) [2]. Éste es el primer estándar de programación verdaderamente multiplataforma para computación en arquitecturas heterogéneas. OpenCL es la alter-

[1]Universidad de Málaga, Andalucía Tech, Dept. of Computer Architecture, Spain. e-mail: {fmlopez, jromero, avillegas, angeles, andres, asenjo}@ac.uma.es.

nativa libre a tecnologías propietarias como CUDA de NVIDIA.

Implementar programas que exploten a la vez CPU y GPU, utilizando OpenCL, puede convertirse en una tarea difícil. Gracias a TBB [1], [3], [4] muchas de estas dificultades desaparecen ya que hace unos años TBB se extendió con un nuevo módulo (*factory*) para poder implementar grafos incluyendo código OpenCL 1.2 [5]. Este nuevo *factory* implementa un nuevo tipo de nodo para Flow Graph, llamado `opencl_node`, el cual permite descargar código a la GPU de una forma mucho más amigable que haciendo uso de OpenCL estándar, y así poder utilizarla como unidad computacional adicional [1], [6]. El `opencl_node` abstrae al programador de los detalles de implementación OpenCL de bajo nivel que antes eran imprescindibles para poder lanzar un código a la GPU.

El objetivo principal de este artículo es hacer una exposición detallada de las nuevas funciones implementadas en el nuevo *factory* para OpenCL 2.0 que hemos desarrollado. Además, evaluaremos el impacto en el rendimiento de esta nueva implementación del *factory* usando el algoritmo ViVid [7] como caso de estudio. ViVid es un código de procesado de *frames* de vídeo que encaja perfectamente con el paradigma de flujo de datos heterogéneo, ya que debemos procesar un *stream* de *frames* en distintas etapas (nodos), algunas de las cuales mapean mejor en una arquitectura GPU.

El artículo tiene la estructura que se indica a continuación. En la Sección II se resumen las ventajas de OpenCL 2.0. En la Sección III profundizamos en el uso y desarrollo del nuevo *factory* OpenCL 2.0 de TBB. Seguidamente, la Sección IV introduce el algoritmo ViVid y resume su implementación con Flow Graph. Por último, se presentan los resultados experimentales en la Sección V para acabar con las conclusiones y trabajo futuro en la Sección VI.

II. Fundamentos

Como ya hemos avanzado, la implementación del nodo OpenCL se lleva a cabo en un nuevo *factory*. En concreto, el *factory* está implementado en el fichero cabecera *flow_graph_opencl_node.h* dentro del directorio de TBB. En éste están definidos todos los objetos que nos facilitan un nivel de abstracción sobre el lenguaje OpenCL, como son: `opencl_node`, `opencl_program`, `opencl_factory`, `opencl_buffer`, `opencl_device` y muchas otras más funciones para conseguir el correcto funcionamiento del nodo OpenCL dentro del sistema de Flow Graph.

Realmente, el diseño de este *factory* está construido sobre otro nodo mucho más enraizado en Flow Graph, el `streaming_node`. La justificación de esta arquitectura *software* en la que encontramos una capa de más bajo nivel (`streaming_node`) que da servicio a la capa *factory* es facilitar a los desarrolladores la implementación de nodos orientados a diferentes lenguajes de programación mediante la creación de nuevos *factories*. El `streaming_node` facilita el uso de modelos de programación para aceleradores, abstrayendo lo que todos los aceleradores tienen en común. Por tanto, `streaming_node` implementa las clases que permiten el envío de datos y *kernels* a diferentes dispositivos a través de un sistema de gestión de colas. Este nodo facilita la interfaz necesaria para integrar fácilmente estos modelos de programación en *Flow Graph*, permitiendo a los desarrolladores usar el grafo para coordinar funciones que se ejecuten en diferentes dispositivos en una arquitectura heterogénea.

En el *factory* de OpenCL están definidos todos los detalles de bajo nivel del modelo, que hasta ahora sólo incluía herramientas de la versión OpenCL 1.2. Añadir soporte para OpenCL 2.0, o por ejemplo CUDA, implica desarrollar el *factory* correspondiente, pero la funcionalidad que proporciona el `streaming_node` es común y encapsula la integración de un `opencl_node`, o un futuro `cuda_node`, con el resto de nodos de Flow Graph.

A. OpenCL 2.0

Con el lanzamiento de OpenCL 2.0 [8] se introdujeron nuevas características que facilitan la colaboración entre CPU y GPU a la hora de procesar datos de forma heterogénea. Algunas de estas características son [9]:

- *Shared Virtual Memory* (SVM). Encontramos de dos tipos: *Coarse Grained* y *Fine Grained*. Éstas son el objeto de este trabajo y las desarrollaremos más adelante.
- *Pipes*. Objetos para gestionar memoria que almacenan datos con una política FIFO. Estos objetos no son accesibles desde el *host*.
- Creación de «colas» en los dispositivos para el encolado de *kernels*. Esto se consigue llamando a *clSetKernelArg*, indicando que el argumento que debe utilizar es de tipo *queue_t* y proporcionándole un puntero a un *device queue*, como se explica en el apartado 5.9.2 de [8].
- Soporte para nuevas operaciones con imágenes; sincronización y operaciones tipo *fence* (operaciones de sincronización) para permitir a los *kernels* leer y escribir imágenes de forma individual.

La característica que nos resulta especialmente relevante es el soporte de *fine-grained* SVM [8], que permite al *host* (CPU), y a los *kernels* ejecutándose en los dispositivos aceleradores, compartir estructuras de datos complejas, como árboles o listas. Además, elimina la necesidad de copiar datos entre el *host* y el *device*, lo que simplifica considerablemente la programación con OpenCL y puede mejorar el rendimiento.

OpenCL 2.0 también soporta *coarse-grained* SVM, pero en este caso la coherencia caché entre los dispositivos no está garantizada por *hardware*. Por el contrario, es necesario invocar operaciones de tipo *map/unmap* que provocan el vaciado de las cachés de CPU o GPU y así reconciliar la visión de los da-

tos entre los dispositivos. Por tanto, este paradigma, *coarse-grained* SVM, es equivalente al conocido como *Zero-Copy-Buffer* existente en OpenCL 1.2, donde tampoco hay que copiar datos entre la CPU y la GPU, pero sólo se garantiza la coherencia en los puntos del código en los que se ejecuten las operaciones de *map/unmap*.

A.1 *Fine Grained* SVM

Podemos encontrar soporte para *fine-grained* SVM en muchas plataformas en las que la CPU y la GPU están integradas en el mismo chip. Con esta funcionalidad es posible reservar una región de memoria que no sólo sea visible al mismo tiempo desde la CPU y la GPU, sino que, además, la coherencia caché o incluso la atomicidad de las operaciones se mantenga por mecanismos *hardware*. El uso de SVM depende de si nuestro dispositivo soporta operaciones *SVM atomics* o no:

- Si las operaciones *SVM atomics* están soportadas se garantiza que el *host* y *device* pueden, concurrentemente, leer y actualizar la misma posición de memoria de forma atómica.
- Por el contrario, si las operaciones *SVM atomic* no están soportadas, *host* y *device* pueden leer la misma posición de memoria, pero no se garantiza la ausencia de condiciones de carrera a no ser que las posiciones de memoria sean distintas. Aún sin soporte *SVM atomic*, *fine-grained* SVM implementa coherencia caché entre la CPU y la GPU lo que permite que la GPU «vea» lo que la CPU ha escrito, y viceversa, sin necesidad de puntos de sincronización que sí son necesarios con *coarse-grained* SVM y que se implementan mediante operaciones de *map/unmap*.

Insistimos por tanto en que la consistencia de memoria está garantizada a nivel de *hardware* sin necesidad de llamar a métodos para el mapeo de memoria, como son *clEnqueueSVMMap* y *clEnqueueSVMUnmap* o *clEnqueueMapBuffer* y *clEnqueueUnmapMemObject* si lo que tenemos es un *buffer* de tipo *cl_mem* que usa un puntero SVM.

A modo ilustrativo, describimos a continuación un caso de uso que es posible implementar cundo tenemos soporte *SVM atomics* y *fine-grained* SVM, pero que no sería factible si sólo disponemos de *coarse-grained* SVM. Supongamos que necesitamos que el *host* y un grupo de dispositivos aceleradores puedan acceder y actualizar simultáneamente una estructura compartida en forma de cola de tareas. Esta configuración permitiría que tanto el *host* como los aceleradores tengan acceso, en exclusión mutua, a una cola de trabajos pendientes. El *host* puede utilizar operaciones atómicas para insertar nuevos trabajos pendientes en la cola a la vez que los dispositivos usan una operación atómica para acceder a esas tareas y extraerlas de la cola para su procesado.

Estas ventajas fueron las que nos motivaron a implementar el *factory* con soporte OpenCL 2.0 tal y como comentaremos en la Sección III.

B. *Driver Intel para OpenCL*

Aunque el objeto de este trabajo se centra en implementar y sacar un mayor partido del *factory* OpenCL 2.0 para Intel TBB, uno de los mayores desafíos fue instalar el *driver* OpenCL en nuestras máquinas.

Hemos hablado de las características de *fine-grained* SVM, pero para conseguir que esto funcione necesitamos una correcta comunicación con los dispositivos. Si accedemos al *git* del *driver* de OpenCL para arquitecturas heterogéneas de Intel (`https://github.com/intel/compute-runtime`), en el apartado *releases* de la guía de instalación podemos leer que "*Fine grained SVM is not supported in this release*" para todas y cada una de las versiones del *driver* que se han lanzado. Esto comenzó a suceder a raíz de una actualización del *kernel Linux*, donde ya no se daba soporte para la coherencia en memoria de los datos. Debido a esto, si seguíamos todas las instrucciones de la guía de instalación, sólo conseguíamos tener soporte para *coarse-grained* SVM, y por tanto, no nos dejaba probar satisfactoriamente todas las mejoras introducidas gracias al soporte *fine-grained* SVM. Si utilizábamos *clinfo* para mostrar las capacidades OpenCL de nuestro chip, obteníamos:

```
Shared Virtual Memory (SVM) capabilities
Coarse-grained buffer sharing           Yes
Fine-grained buffer sharing             No
Fine-grained system sharing             No
Atomics                                 No
```

Durante este trabajo, conseguimos corregir los errores que había en los ficheros fuente del *driver* y dar soporte SVM a nuestros equipos. En futuras fechas se prevé publicar una guía, adjunta a un TFG, que se creó para solucionar este problema. Mientras tanto, se puede seguir el hilo de alguno de los comentarios que se dejó a los moderadores del repositorio con la misión de resolver los fallos: `https://github.com/intel/compute-runtime/issues/134`

III. Nodo OpenCL de Flow Graph

Ya hemos introducido el *factory* y tenemos una idea global del problema que hay que afrontar. Lo siguiente es identificar las partes susceptibles de modificación para integrar el uso de OpenCL 2.0 y SVM. Para esto hay que profundizar en cada una de las clases y ver hasta donde llegan las dependencias de cada una de ellas.

Para facilitar la comprensión del *factory* de OpenCL vamos a presentar primero el código que debería escribir un usuario para usarlo. La Figura 2 incluye un código casi completo que ejecuta la operación vectorial $C = A+B$, en la GPU. Los vectores A, B, C son de tamaño `vsize` (inicializado en la línea 1) y contienen datos de tipo `float`. En la línea 6 se declaran los tres vectores de tamaño `vsize` y de tipo `buffer_f` que está definido en la línea anterior como `opencl_buffer<float>`. A continuación, se inicializan los vectores A y B con la secuencia {0, 1, 2, ...} gracias al algoritmo `iota` de la Standar Template Library, STL.

```
1  size_t vsize = (argc>1) ? atoi(argv[1]) : 1000; //Problem size
2
3  tbb::flow::graph g; // objeto flow graph
4
5  using buffer_f = tbb::flow::opencl_buffer<cl_float>;
6  buffer_f A{vsize}, B{vsize}, C{vsize}; // Vectores para C=A+B
7
8  // Inicializa A y B
9  std::iota(A.begin(), A.end(), 0); // 0, 1, 2, 3, ...
10 std::iota(B.begin(), B.end(), 0); // 0, 1, 2, 3, ...
11
12 //GPU node:
13 gpu_device_selector selector;
14 tbb::flow::opencl_program<> program{std::string{"kernel.cl"}};
15 tbb::flow::opencl_node<std::tuple<tbb::flow::opencl_range>>
16     gpu_node{g, program.get_kernel("kernel_add"), selector}; // C=A+B en GPU
17
18 gpu_node.set_range(tbb::flow::port_ref<0>);
19 gpu_node.set_args(A, B, C);
20 tbb::flow::input_port<0>(gpu_node).try_put(
21   tbb::flow::opencl_range({static_cast<int>(vsize)}));
22
23 g.wait_for_all();
24 // Comprueba si el nodo GPU ha completado bien su trabajo
25 std::vector<float> CGold(vsize); // Array de referencia con C=A+B en CPU
26 std::transform(A.begin(), A.end(), B.begin(), CGold.begin(), std::plus<float>());
27 if (! std::equal (C.begin(), C.end(), CGold.begin())) //Comprueba
28   std::cout << "Errors in the GPU computation.\n";
```

Fig. 2. Implementación de un grafo que incluye un `opencl_node` que ejecuta $C = A + B$ en la GPU

La configuración del nodo OpenCL se implementa en sólo 5 líneas de código. En la línea 13 se crea un objeto de la clase `gpu_device_selector` que no explicamos en el ejemplo pero que permite seleccionar el dispositivo OpenCL que va a ejecutar el *kernel* en caso de que haya varios (Capítulo 19 de [1]). La línea 14 construye el objeto `program` inicializado con el fichero `kernel.cl` que debe contener el *kernel* a ejecutar en el acelerador. En este ejemplo, el fichero `kernel.cl` únicamente contiene las líneas que codifican la operación $C = A + B$ en OpenCL:

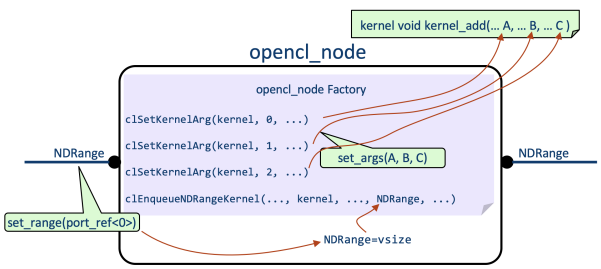


Fig. 3. Interior del `opencl_node` de la Figura 2

```
kernel void kernel_add(global float* A,
                       global float* B,
                       global float* C){
  int i = get_global_id(0);
  C[i] = A[i] + B[i];
}
```

Quizás la línea 16 sea la más importante de todas ya que crea el objeto `gpu_node` de tipo `opencl_node`. Los argumentos del template:

```
opencl_node<std::tuple<tbb::flow::
    opencl_range>>
```

especifican mediante una `std::tuple<>` el número y el tipo de datos para cada puerto del nodo. En este caso el nodo sólo tiene un puerto de tipo `tbb::flow::opencl_range` por el que llegará el `NDRange` que especifica el espacio de iteraciones que recorre el *kernel*. La Figura 3 ilustra el nodo `opencl_node` que estamos construyendo con un único puerto de entrada y de salida, como acabamos de explicar.

El constructor con el que creamos el objeto `gpu_node` (seguimos en la línea 16) contiene tres argumentos: i) el grafo al que pertenece el nodo, `g`; ii) el nombre del *kernel* que el nodo ha de ejecutar, `kernel_add`; y iii) el objeto `selector` que hemos inicializado en la línea 13. La función miembro `set_range` de `gpu_node` se usa en la línea 18 para especificar que por el puerto 0 de este nodo es por donde llegará la información con el `NDRange`. De forma similar, la función miembro `set_args` de la línea 19 mapea los *buffers* A, B y C en los correspondientes argumentos del *kernel*. En la Figura 3 se aprecia como esa llamada provocará que el nodo OpenCL invoque tres veces a la función `clSetKernelArg` de la librería OpenCL.

Con las cinco líneas comentadas, el nodo `gpu_node` está configurado, pero no ejecutará el *kernel* hasta que le llegue un mensaje por el puerto de entrada. En este caso, la llamada a `try_put` de la línea 20 es la encargada de disparar la ejecución del nodo. El mensaje que se envía con `try_put` es un `opencl_range` de una dimensión e inicializado con el tamaño de los vectores, `vsize`. En la Figura 3 se muestra que esto se traducirá internamente por una llamada a la función `clEnqueueNDRangeKernel` de OpenCL, la cual activa la ejecución del *kernel* en la GPU. La llamada se configura adecuadamente con el *kernel* especificado al construir `gpu_node` en la línea 16 y con el `NDRange` que acabamos de describir.

Las últimas líneas de la Figura 2 sirven para esperar a que el nodo termine la computación (línea 23)

```
1  ...
2  gpu_device_selector selector;
3  using factory_t = opencl_factory<gpu_device_selector>;
4  factory_t my_factory( CL_MEM_READ_WRITE |
5                        CL_MEM_SVM_FINE_GRAIN_BUFFER |
6                        CL_MEM_SVM_ATOMICS );
7
8  using buffer_f = tbb::flow::opencl_buffer_svm<cl_float, factory_t>;
9  buffer_f A{my_factory, vsize}, B{my_factory, vsize}, C{my_factory, vsize};
10 // Initialize A and B
11 ...
12 //GPU node:
13 tbb::flow::opencl_program<factory_t> program{my_factory, std::string{"kernel.cl"}};
14 tbb::flow::opencl_node<std::tuple<tbb::flow::opencl_range,buffer_f>, queueing, factory_t>
15           gpu_node{g, program.get_kernel("kernel_add"), selector, my_factory};
16 ...
```

Fig. 4. Modificaciones necesarias en el ejemplo de la Figura 2 para usar *buffers* SVM

y comprobar que la GPU ha realizado correctamente su trabajo (comparando la salida de la GPU con una versión de referencia computada en la CPU).

A. Uso del nuevo factory OpenCL 2.0

Antes de discutir los cambios realizados en el *factory* (mediante modificaciones en el fichero cabecera `flow_graph_opencl_node.h`), vamos a describir qué implicaciones tiene su uso de cara al programador. En primer lugar hay que decir que las modificaciones para dar soporte a OpenCL 2.0 son transparentes al programador que no necesite SVM. Esto significa que el código de la Figura 2 es totalmente funcional con el nuevo *factory*, pero si no añadimos nada se usará el mecanismo de gestión de datos basado en *coarse-grained* SVM con *map/unmap* propio de de OpenCL 1.2 (es decir, no explotará *fine-grained* SVM).

Si el usuario quiere aprovechar las características *fine-grained* SVM del nuevo *factory*, el usuario debe modificar ligeramente su implementación configurando el *factory* y creando un nuevo tipo de *buffers*. En la Figura 4 se muestran las diferencias de implementación con respecto al ejemplo de la Figura 2.

En primer lugar vemos que en la línea 4 se crea un nuevo objeto `my_factory` del tipo `factory_t` definido en la línea anterior y configurado con los flags SVM que el código va a necesitar. En este caso, estamos especificando que los *buffers* serán de lectura/escritura y deben soportar *fine-grained* SVM así como *SVM atomics*.

En la línea 9 vemos como se declaran ahora los tres vectores, A, B y C, donde notamos dos diferencias: i) el constructor incluye el argumento `my_factory` que hemos creado anteriormente; y ii) el tipo `buffer_f` no es ya un alias de `opencl_buffer` (como en la Figura 2) sino del nuevo tipo `opencl_buffer_svm` que hemos implementado en el nuevo *factory*.

También hay ligeras diferencias en la declaración del objeto `program` y `gpu_node` (líneas 13 y 15), que básicamente consisten en incluir el tipo `factory_t` como argumento del template y `my_factory` como argumento del constructor. El resto de las líneas del ejemplo de la Figura 2 permanecen invariables y se han omitido mediante puntos suspensivos por brevedad.

B. Relación con el módulo `streaming_node`

En sentido estricto, un *factory* (por ejemplo, el `opencl_node`) no es más que una instanciación de un `streaming_node` al que se dota de las particularidades del modelo de programación específico al que se quiere dar soporte (en este caso, OpenCL). Aunque por motivos de espacio no podemos explicar todos los detalles de las clases soportadas por `streaming_node`, la Figura 5 nos ayudará a resumir los aspectos más importantes y que necesitamos cubrir para luego entender las modificaciones que implementamos en el `opencl_node`. Para una información más detallada, remitimos al lector a la documentación disponible en [10].

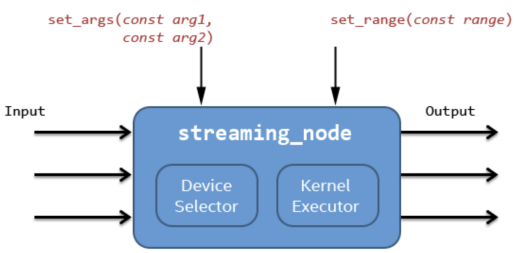


Fig. 5. Diagrama del `streaming_node`

En la Figura 5 se resume la estructura interna de un `streaming_node`. Las características más importantes son:

- Un `streaming_node` recibe los datos de entrada mediante *input ports*. Éstos se pueden mapear en argumentos del *kernel*, en el `NDRange` del *kernel* o sencillamente salir del nodo sin modificación alguna (*pass through*).
- Por cada puerto de entrada existe el puerto de salida que, en caso de conectarse con otro nodo del grafo, propaga el mensaje de entrada (modificado o no por el acelerador) al siguiente nodo.
- Contiene una clase `DeviceSelector` que se puede usar por defecto, o configurar por el programador, para seleccionar el dispositivo acelerador que ejecutará el *kernel*.
- Contiene una clase `KernelExecutor` encargada de gestionar los aspectos independientes del acelerador que tengan que ver con el lanzamiento

del *kernel*.
- Da soporte a las funciones miembro `set_args` y `set_range` que ya se han descrito previamente en relación con la Figura 2.

C. Implementación del factory opencl_node 2.0

Una vez descritos los fundamentos necesarios, pasamos a resumir los cambios que se han implementado en el nuevo *factory*. La primera modificación consiste en dar soporte para poder indicar los *flags* de control de sincronización para SVM (tal y como se hace en la línea 4 de la Figura 4). Para ello se han implementado dos constructores de la clase `opencl_factory` tal y como se detalla en la Figura 6.

```
1 opencl_factory() : my_flags(
      CL_MEM_READ_WRITE) {}
2 opencl_factory(cl_svm_mem_flags flags) :
       my_flags(flags) {}
```

Fig. 6. Constructor de la clase `opencl_factory`

La clase contiene ahora una nueva variable miembro, `my_flags`, de tipo `cl_svm_mem_flags`. La primera línea de la Figura 6 describe el constructor por defecto (el que se llama si creamos un `opencl_factory` sin pasar ningún argumento) donde `my_flags` se inicializa sólo con `CL_MEM_READ_WRITE`. Si por el contrario se especifican `flags` en el argumento, estos se usan para inicializar `my_flags`.

D. Nueva clase *opencl_buffer_svm*

La clase `opencl_buffer` del *factory* original (usada en la Figura 2) se apoya en otra, `opencl_buffer_impl`, cuya clase base es `opencl_memory`. Estas clases realizan las operaciones para crear los *buffers* en regiones de memoria accesibles por la CPU y el acelerador, además de ejecutar las correspondientes operaciones de *map/unmap* para asegurar la coherencia necesaria en el modelo *Zero-Copy-Buffer* de OpenCL 1.2. Con objeto de dar soporte al OpenCL 2.0 manteniendo la compatibilidad descendente con 1.2 hemos creado clases análogas a las anteriores con el prefijo `_svm`: `opencl_buffer_svm`, `opencl_buffer_svm_impl` y `opencl_memory_svm`.

La creación de un `opencl_buffer_svm` se realiza en el constructor de la clase, el cual llama a la función de OpenCL `clCreateBuffer`. Por comparación con la implementación original, la Figura 7 muestra la llamada a `clCreateBuffer` del constructor de la clase `opencl_buffer`. Se aprecia que esa llamada devuelve un `cl_mem` o región de memoria de tipo OpenCL que es visible por la GPU, pero también por la CPU ya que usamos el flag `CL_MEM_ALLOC_HOST_PTR` que habilita el uso de las funciones *map/unmap* para poder obtener el puntero a la misma región de memoria visible ahora desde la CPU.

En la Figura 8 mostramos la modificación implementada en el constructor de `opencl_buffer_svm`. Por un lado vemos como reservamos la memoria

```
3 this->my_cl_mem = clCreateBuffer(
4         this->my_factory->context(),
5         CL_MEM_ALLOC_HOST_PTR,
6         size,
7         NULL,
8         &err );
```

Fig. 7. Constructor `opencl_buffer` (OpenCL 1.2)

```
9 this->my_ptr = (void *) clSVMAlloc (
10        this->my_factory->context(),
11        this->my_factory->my_flags,
12        size,
13        0 );
14
15 this->my_cl_mem = clCreateBuffer(
16        this->my_factory->context(),
17        CL_MEM_USE_HOST_PTR,
18        size,
19        this->my_ptr,
20        &err );
```

Fig. 8. Constructor `opencl_buffer_svm` (OpenCL 2.0)

con una primera llamada a `clSVMAlloc`, función de OpenCL 2.0 que devuelve un puntero (`void`) a una región de memoria SVM visible tanto por la CPU como por la GPU. Sin embargo, realizamos también una llamada a `clCreateBuffer`, ahora pasando el flag `CL_MEM_USE_HOST_PTR`, el cual modifica el comportamiento de `clCreateBuffer` que ahora no reserva memoria, pero si devuelve el `cl_mem` correspondiente al puntero `void` obtenido anteriormente. De esta forma `my_ptr` y `my_cl_mem` apuntan al mismo sitio. Esta estrategia nos permite usar el alias `my_cl_mem` para no tener que modificar la parte de el `factory` original que inicializa los argumentos del *kernel* mediante la llamada OpenCL `clSetKernelArg`. Si por el contrario sólo inicializamos el puntero SVM `my_ptr`, el paso de argumentos al *kernel* debe hacerse mediante llamadas a `clSetKernelArgSVMPointer`, complicando innecesariamente el nuevo *factory*.

El destructor de la nueva clase también necesita una ligera modificación. Para el destructor de un `opencl_buffer` basta con llamar a la función OpenCL `clReleaseMemObject`. Sin embargo, para destruir un `opencl_buffer_svm` además de la llamada anterior que deshace el `clCreateBuffer` del constructor, también hay que llamar a `clSVMFree` para liberar la memoria reservada en el `clSVMAlloc`, teniendo cuidado de conservar este orden: primero eliminamos el *buffer* y posteriormente liberamos memoria.

E. Simplificaciones habilitadas por OpenCL 2.0

Gracias a OpenCL 2.0 podemos eliminar gran parte de la complejidad implementada en el *factory* original basado en OpenCL 1.2. Por ejemplo, en este último era necesario implementar las operaciones de *map/unmap* para garantizar el acceso coherente a los *buffers* procesados en la GPU. En el *factory* original estas operaciones se implementan en las funciones `receive` y `send`:

- **receive**: Esta función se invoca cuando un `opencl_node` «recibe» un mensaje de tipo `opencl_buffer` que está asociado a uno de los argumentos del *kernel*. Esto implica que los datos, potencialmente modificados en la CPU, van a ser usados ahora en la GPU. Aunque gracias al *Zero-Copy-Buffer* no hay que copiar físicamente datos de un dispositivo a otro (el *buffer* es único en memoria), sí hay que asegurarse de que la CPU hace un *flush* de los datos que tenga cacheados y no estén actualizados en el *buffer*. Por tanto antes de lanzar el *kernel* que va a acceder al *buffer* desde la GPU, esta función `receive` llama a la función `map_memory` que a su vez invoca a la función OpenCL `enqueue_map_buffer`. Esta última provoca la sincronización necesaria para que la GPU vea las modificaciones realizadas en CPU.
- **send**: Esta función se invoca cuando un `opencl_node` «envía» un mensaje de tipo `opencl_buffer` al siguiente nodo del grafo. Ahora es la GPU la que pude tener datos modificados en sus cachés y es necesario hacer un *flush* que garantice la coherencia con la vista que la CPU tiene de los datos. Por tanto, esta función `send`, termina llamando a la función `enqueue_unmap_buffer` de OpenCL para conseguir el efecto deseado.

Como vemos, la coherencia en OpenCL 1.2 se garantiza sólo tras el encolado de ciertas funciones en las colas OpenCL que provocan la sincronización de los datos en memoria global. Este es el mismo mecanismo que se debe implementar en OpenCL 2.0 cuando sólo tenemos disponible *coarse-grained* SVM. Sin embargo, si disponemos de *fine-grained* SVM, la coherencia entre las cachés de CPU y GPU está garantizada por *hardware* y las operaciones de *map/unmap* son innecesarias. De este modo, las funciones `send` y `receive` no tienen nada que hacer (aunque se mantienen con el cuerpo vacío para permitir que las clases `opencl_buffer` y `opencl_buffer_svm` coexistan en el mismo *factory*).

La CPU también puede acceder a datos potencialmente modificados por la GPU usando los métodos `begin()` o `data()`, tal y como ocurre en la línea 27 de la Figura 2 con el *buffer* `C`. Esto provoca en el *factory* la llamada a `get_host_ptr()`, definida en `opencl_memory`. Para un `opencl_buffer` esta función debe invocar a `enqueue_map_buffer` y posteriormente devolver el puntero. Sin embargo, en el caso de que estemos utilizando *fine-grained* SVM con `opencl_buffer_svm` del nuevo *factory*, obviamos el *unmap* y sencillamente devolvemos el puntero tal y como se muestra en la Figura 9.

```
21 void* get_host_ptr() const {
22         __TBB_ASSERT( my_ptr, NULL );
23         return my_ptr;
24 }
```

Fig. 9. Implementación de `get_host_ptr` para un *buffer* SVM

IV. Caso de estudio: ViVid

Con objeto de validar el nuevo *factory* desarrollado, usaremos una aplicación de procesado de imágenes en *pipeline* llamada ViVid [7], [11] (*The Video Processing Library*). Esta aplicación ya fue desarrollada para arquitecturas heterogéneas usando TBB Flow Graph y `opencl_node` tal y como se describe en [12]. Por completitud, resumimos a continuación los aspectos relevantes y necesarios para comprender los resultados experimentales. ViVid es una aplicación con patrón *pipeline* de 3 etapas que emplea una técnica de «ventana deslizable» para la detección de objetos en *frames* de vídeo. En la Figura 10 se puede ver la distribución de las 3 etapas en las que se divide el algoritmo.

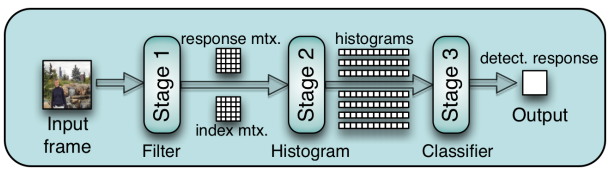


Fig. 10. Algoritmo ViVid.

Tal y como muestra la Figura 11, la implementación heterogénea permite que las tres etapas del *pipeline* se ejecuten en CPU y/o en GPU. Usando una palabra binaria de tres bits, se configura la topología del *pipeline* de forma que se habilite el uso de la GPU en las etapas que tengan un 1 en la posición correspondiente. Por ejemplo, la topología 101 permite que algunos de los *frames* de vídeo se procesen en la GPU, pero sólo para las etapas 1 y 3, mientras que todos los *frames* son procesados por la etapa 2 en la CPU.

La implementación de ViVid descrita en [12] usa Flow Graph y la versión de `opencl_node` basada en el *factory* original con OpenCL 1.2. La Figura 12 muestra el grafo heterogéneo donde cada *frame* de vídeo pueden seguir un camino en el que todas las etapas se procesan en la CPU o algunas de ellas en GPU. Para gestionar la disponibilidad de los recursos (*cores* y GPU) se usa una estrategia basada en «*tokens*». Si queremos explotar 4 *cores* y una GPU, configuramos la aplicación con cuatro *tokens* de CPU y uno de GPU. Los cinco *tokens* circulan por el grafo, cada uno de ellos tomando un *frame* y procesándolo donde corresponda. Por ejemplo, un *token* de GPU en una configuración 101 acompaña un *frame* de forma que las etapas 1 y 3 se ejecuten en la GPU y la etapa 2 en CPU. Un *token* de CPU acompaña a un *frame* de forma que todas sus etapas se procesan en CPU. Para una descripción más detallada dirigimos al lector a la Sección III de [12].

Modificar la versión existente para que use el nuevo *factory* basado en OpenCL 2.0 fue trivial ya que sólo implicó realizar cambios similares a los que describimos en la Figura 4.

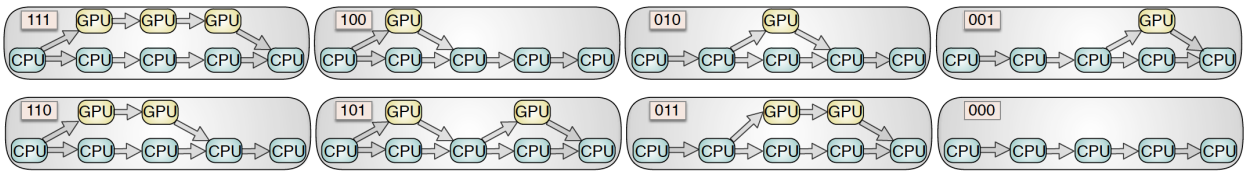


Fig. 11. Pipeline ViVid.

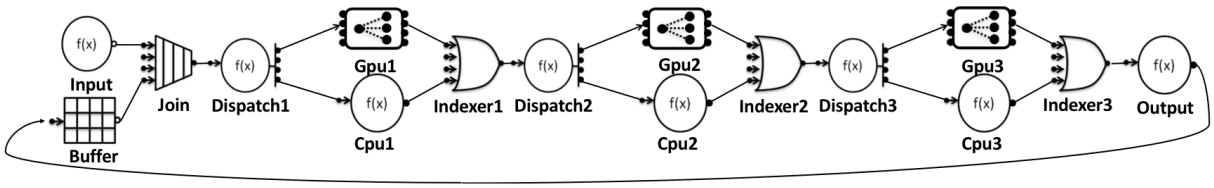


Fig. 12. FlowGraph de ViVid.

V. Resultados experimentales

En este apartado se van a mostrar los resultados obtenidos al ejecutar las dos versiones de ViVid. A la versión que usa el *factory* original basado en OpenCL 1.2 la llamaremos «NoSVM» mientras que a la versión que explota el nuevo *factory* OpenCL 2.0 la llamaremos «SVM».

El *hardware* sobre el que se han realizado las pruebas de rendimientos está basado en una plataforma Intel de novena generación con arquitectura Coffee Lake y las siguientes características:

- CPU: 8 cores Intel i9-9900K a 3.60GHz con 8x256KB de caché L2 y 16MB de caché L3 (compartida entre los 8 cores)
- GPU: Intel(R) UHD Graphics 630 a 350-1200MHz.
- RAM: 32GB
- OS: Ubuntu 18.04.2 LTS, kernel 5.0.0-custom adaptado para soportar *fine-grained* SVM con el driver de OpenCL 2.1 NEO 19.08.0.
- TBB: Versión 2019 update 3 incluido en Intel System Studio 2019
- Compilador: Intel icc 2019

Además del software especificado, también hemos hecho uso de la librería Catapult [13] que nos ofrece funciones para visualizar la traza de ejecución de los *frames* procesados en ViVid. Mas adelante veremos que estas trazas permiten explicar visualmente el problema del desbalanceo entre la CPU y GPU y cómo lo resolvemos en nuestra implementación.

A. Estudio previo

Para el estudio experimental alimentaremos ViVid con dos resoluciones de vídeo. La de baja resolución, SD, tiene *frames* de 414x600 pixels, mientras que la de alta resolución, HD, tiene *frames* de 1080x1980. Los *frames* de baja resolución ejercen menos presión sobre el sistema de memoria, pero tener en vuelo varios *frames* de alta resolución (por ejemplo, 8 siendo procesados en los 8 *cores* y un 9º siendo procesado en GPU) va a suponer una mayor demanda de ancho de banda de memoria. La longitud del vídeo SD es de 7000 *frames* y la del vídeo HD de 1400 *frames*. Estos números de *frames* se han elegido de forma que la ejecución de ViVid usando únicamente la GPU consuma alrededor de 30 segundos.

En un primer estudio, hemos buscado cuál es la configuración óptima del *pipeline* heterogéneo que reporte un mayor *throughput* para vídeo SD y HD. Los resultados obtenidos en nuestra plataforma concluyen que para imágenes SD, la configuración más rápida es la 101 en la que la etapa 2 siempre se ejecuta en CPU (ver Figura 11). Sin embargo, para imágenes HD obtenemos mejores resultados con la configuración 111 en la que un *token* de GPU procesa un *frame* de vídeo en la GPU para todas las etapas. Este comportamiento es explicable ya que, para imágenes SD, el volumen de trabajo a descargar a la GPU es de grano muy fino para la etapa 2, no ocupa totalmente los recursos disponibles en la GPU y por tanto no amortiza los costes del lanzamiento del *kernel* y la correspondiente sincronización con la CPU.

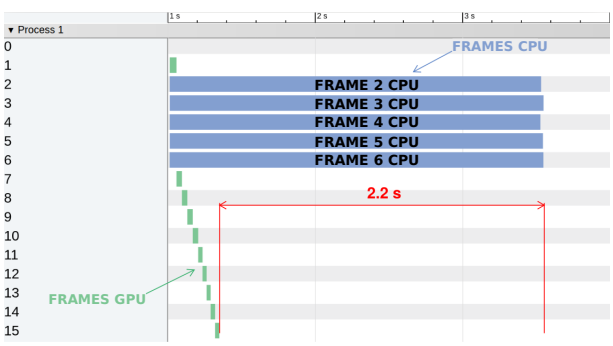


Fig. 13. Exceso de tiempo

Otro parámetro a configurar en la ejecución de ViVid tiene que ver con la gestión del desbalanceo. El problema que queremos evitar se ilustra en la Figura 13, donde vemos una traza de ejecución de 15 *frames* de vídeo con ViVid en configuración 111, 1 *token* de GPU y 5 *tokens* de CPU. El primer *token* de GPU ejecuta el primer *frame* en la GPU y los 5 *tokens* de CPU acompañan a 5 *frames* por todas las etapas del

pipeline en CPU requiriendo más de 3.5seg. En ese tiempo, el *token* de GPU se puede reciclar (recorrer el *pipeline* y volver al principio a recoger otro *frame*) 9 veces y terminar de procesar los 9 *frames* restantes en la GPU. Sin más *frames* disponibles, la GPU se queda ociosa 2.2 seg esperando a que los *cores* terminen de procesar su parte.

Para evitar esta situación, ViVid acepta un parámetro que llamamos `cpu_limit` que especifica el porcentaje máximo del número de *frames* que se pueden procesar en CPU. Cuando el contador de *frames* detecta que hemos alcanzado el `cpu_limit` desconecta las CPUs y redirige todos los *frames* restantes a la GPU. El parámetro `cpu_limit` se puede computar automáticamente midiendo el tiempo de ejecución de los primeros *frames* que se procesan en CPU y en GPU. La tabla I, indica el tiempo en milisegundos por *frame* para vídeo SD y HD en CPU y GPU para las configuraciones 101 (SD) y 111 (HD) cuando trabajan los 8 *cores* y la GPU.

TABLA I
TIEMPOS MEDIOS $[ms/frame]$

Configuración		*CPU*	*GPU*
SD	NoSVM	268	4.14
	SVM	255	4.15
HD	NoSVM	2295	20.18
	SVM	2204	19.83

De la información disponible en la tabla I vemos que la GPU es entre 61x y 63x más rápida que un *core* en SD y entre 111x y 113x más rápida que un *core* en HD (cuando los *cores* 8 están computando). Con esta información podemos calcular un valor estimado para `cpu_limit`, que va a depender también del número de *frames* del vídeo. Por ejemplo, para HD y SVM, la GPU es 2204/19.83=111x más rápida que la CPU. Dado que tenemos 1400 *frames* HD y queremos garantizar que los últimos 111 los ejecute la GPU, el valor resultante para `cpu_limit` quedaría como (1400-111)/1400=0.92, que indica que cuando se hayan procesado el 92 % de los *frames* debemos desactivar los *cores* y redirigir todos los *frames* por el camino GPU. Para SD, donde procesamos 7000 *frames* y la velocidad relativa de la GPU sobre la CPU es menor, hemos de parar los *cores* cuando se superen el 99 % de los *frames*.

B. Comparación SVM vs. NoSVM

Las Figuras 14 y 15 presentan el *speedup* de las ejecuciones en GPU y en GPU mas 1 a 8 *cores*, relativo a la ejecución secuencial en un sólo *core* para vídeo SD y HD respectivamente. La línea discontinua azul muestra el *speedup* de la versión NoSVM y la verde la versión SVM. Aunque la GPU es mucho más rápida que un *core* procesando *frames*, la gráfica muestra que ir añadiendo *cores* de CPU acelera linealmente el *speedup* de forma que la ejecución en GPU+8cores es un 10 % en SD y un 7.2 % en HD más rápida que si sólo usamos la GPU.

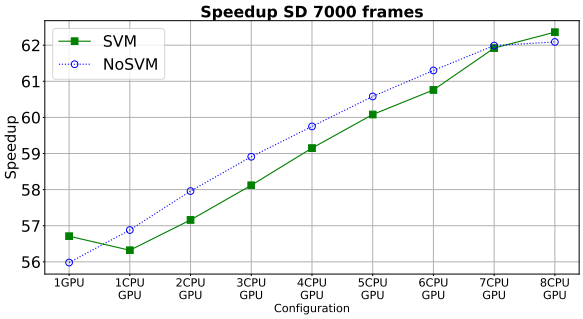


Fig. 14. *Speedup* sobre 1 *core* para SD, NoSVM vs. SVM

En el caso de SD, la implementación NoSVM es ligeramente mejor excepto para los casos 1GPU y GPU+8cores donde la versión SVM es algo más rápida. En esta situación, en la que hay menos localidad que explotar debido a *frames* de menor tamaño, las operaciones de *map/unmap* y las correspondientes operaciones *flush* de caché tienen un menor impacto que el coste de mantener la coherencia por *hardware* (como ocurre en la versión SVM).

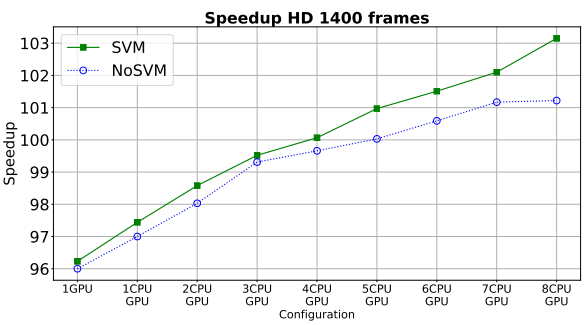


Fig. 15. *Speedup* sobre 1 *core* para HD, NoSVM vs. SVM

Sin embargo, para HD la SVM consistentemente mejora a la versión NoSVM para cualquier configuración heterogénea. En este caso compensa evitar los *flush* de caché implícitos en las operaciones *map/unmap* de la versión NoSVM y gestionar por *hardware* los posibles fallos de coherencia caché en la versión SVM. En un estudio que llevaremos a cabo en breve, validaremos esta afirmación midiendo el número de fallos de caché para las distintas implementaciones.

Aunque la diferencia en tiempo de ejecución para esta aplicación ViVid entre la versión SVM y la NoSVM no sea muy significativa, el soporte SVM puede ser muy relevante en otras aplicaciones ya que permite un grado de sincronización entre la CPU y la GPU a un grano más fino.

C. Comparación con el rendimiento ideal

Por último hemos querido también explorar cómo de lejos están nuestras implementaciones de un hipotético ideal y cuantificar de algún modo hasta que punto el *throughput* combinado de la CPU y la GPU trabajando conjuntamente es mejorable. Para ello hemos medido el *throughput* máximo cuando sólo los *cores* o sólo la GPU procesan *frames*. En este trabajo, consideramos que la suma de esos dos *throughputs*

es el máximo alcanzable. En la Figura 16 ese máximo ideal está representado mediante una línea roja en el 100 %.

En la realidad, cuando todos los *cores* y la GPU trabajan al mismo tiempo procesado diferentes *frames*, la frecuencia de los *cores* y de la GPU ha de reducirse para que no se supere el límite de TDP (*Thermal Design Power*) que en el caso de nuestro procesador es de 95 vatios. Además, el ancho de banda máximo de memoria también está limitado, y si la GPU y la CPU están procesando *frames* al mismo tiempo se alcanzará ese límite. En la Figura 16 representamos con barras el *throughput* real obtenido en la ejecución GPU+8cores para SD y HD en las configuraciones SVM y NoSVM, relativo al *throughput* ideal. Como vemos, para vídeo HD, el *throughput* real está ligéramente más próximo al ideal (entre el 99.4 % y el 97.5 %) que en el caso de procesar vídeo SD (entre el 96.4 % y el 97 %).

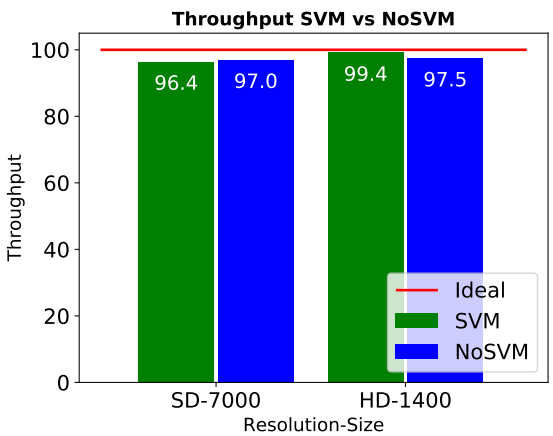


Fig. 16. Porcentaje de *throughput* relativo al ideal

VI. Conclusiones

En este trabajo hemos implementado y evaluado un nuevo *factory* para la clase `opencl_node` de Intel TBB. La versión original proporcionada por Intel soporta OpenCL 1.2 y nuestra contribución ha consistido en extender el soporte a OpenCL 2.0 y así poder habilitar las mejoras introducidas en la última versión. En particular hemos explotado el soporte de SVM, en la modalidad *fine-grained* SVM, en la que la coherencia caché entre la CPU y la GPU se mantiene por *hardware*. En comparación con el soporte original basado en OpenCL 1.2 y el modelo *Zero-Copy-Buffer*, el nuevo soporte evita los puntos de sincronización invocados mediante operaciones *map/unmap* necesarios para vaciar las cachés y conseguir así una visión coherente de los *buffers* usados en CPU y GPU. A cambio, mantener la coherencia por *hardware* también tiene un precio, pero: i) puede ser inferior al que resulta de los *flush* de caché invocados en las operaciones *map/unamp*; y ii) permite implementaciones paralelas y heterogéneas en las que la estructura de datos procesada, tanto en CPU como en GPU, se accede con una menor granularidad y de forma continua.

Los resultados experimentales obtenidos para una implementación heterogénea de la aplicación ViVid concluyen que el *factory* OpenCL 2.0 reporta un rendimiento similar al original implementado por Intel y que se basa en OpenCL 1.2. Además, el *throughput* de la ejecución heterogénea tanto para vídeo SD como HD está por encima del 96 % del ideal.

Como trabajo futuro, a corto plazo validaremos, mediante la instrumentalización del código con Intel PCM, que se producen menos fallos de caché en el nuevo *factory* OpenCL 2.0. También implementaremos otro tipo de aplicaciones en las que la CPU y la GPU tenga una colaboración más estrecha y el soporte SVM tenga un mayor impacto.

Agradecimientos

El presente trabajo ha sido financiado mediante el proyecto TIN2016-80920-R del Plan Nacional de Investigación, y por la Universidad de Málaga (Campus de Excelencia Internacional Andalucía Tech).

Referencias

[1] Michael Voss, Rafael Asenjo, and James Reinders, *Pro TBB: C++ Parallel Programming with Threading Building Blocks*, Apress, 1st edition, 2019, 972 pages.

[2] "OpenCL Overview," `https://www.khronos.org/opencl/`, Accessed: 2019-06-19.

[3] "Tutorial Intel TBB," `http://www.inf.ed.ac.uk/teaching/courses/ppls/TBBtutorial.pdf`, Accessed: 2019-06-19.

[4] "Tutorial Intel TBB Web," `https://www.threadingbuildingblocks.org/intel-tbb-tutorial`, Accessed: 2019-06-19.

[5] Alexei Katranov and Alexey Kukanov, "Intel Threading Building Block (Intel TBB) flow graph as a software infrastructure layer for OpenCL-based computations," pp. 1–3, 04 2016.

[6] Michael Voss, Pablo Reble, and Jackson Marusarz, "CPUs, GPUs, FPGAs: Managing the alphabet soup with Intel Threading Building Blocks," in *Tutorial at PPoPP 2017 Intl. Symp. on Principles and Practice of Parallel Programming*, 2017.

[7] Mert Dikmen, *Visual Detection and Recognition using local features*, Ph.D. thesis, University of Illinois, 2012, Accessed: 2019-06-30.

[8] "The OpenCL Specification," `https://www.khronos.org/registry/OpenCL/specs/opencl-2.0.pdf`, Accessed: 2019-06-19.

[9] David R. Kaeli, Perhaad Mistry, Dana Schaa, and Dong Ping Zhang, *Heterogeneous Computing with OpenCL 2.0*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition, 2015.

[10] "Streaming-node TBB Documentation," `https://software.intel.com/en-us/node/684813`, Accessed: 2019-06-19.

[11] Kurt Fellows, "A comparative study of the effects of parallelization on ARM and Intel based platforms, master thesis," `http://hdl.handle.net/2142/50710`, 2014, Accessed: 2019-06-30.

[12] José Carlos Romero, Alejandro Villegas, Ángeles Navarro, Andrés Rodríguez, and Rafael Asenjo, "Explotando el nuevo módulo OpenCL de Intel TBB," in *XXIX Edición de las Jornadas de Paralelismo, JP'18*, Teruel, Spain, Sept 2018.

[13] "Catapult Tool," `https://github.com/catapult-project/catapult`.

Parallel Image Processing Using a Pure Topological Framework

Fernando Diaz-del-Rio[1], Helena Molina-Abril[2], Pedro Real[2], Pablo Sánchez-Cuevas[1], Antonio Ríos-Navarro[1]

Abstract— **Image processing is a fundamental operation in many real time applications, where lots of parallelism can be extracted. Segmenting the image into different connected components is the most known operations, but there are many others like extracting the region adjacency graph (RAG) of these regions, or searching for features points, being invariant to rotations, scales, brilliant changes, etc. Most of these algorithms part from the basis of Tracing-type approaches or scan/raster methods. This fact necessarily implies a data dependence between the processing of one pixel and the previous one, which prevents using a pure parallel approach. In terms of time complexity, this means that linear order *O(N) (N* being the number of pixels) cannot be cut down. In this paper, we describe a novel approach based on the building of a pure Topological framework, which allows to implement fully parallel algorithms. Concerning topological analysis, a first stage is computed in parallel for every pixel, thus conveying the local neighboring conditions. Then, they are extended in a second parallel stage to the necessary global relations (e.g. to join all the pixels of a connected component). This combinatorial optimization process can be seen as the compression of the whole image to just one pixel. Using this final representation, every region can be related with the rest, which yields to pure topological construction of other image operations. Besides, complex data structures can be avoided: all the processing can be done using matrixes (with the same indexation as the original image) and element-wise operations. The time complexity order of our topological approach for a *m*×*n* pixel image is near *O(log(m+n))*, under the assumption that a processing element exists for each pixel. Results for a multicore processor show very good scalability until the memory bandwidth bottleneck is reached, both for bigger images and for much optimized implementations. The inherent parallelism of our approach points to the direction that even better results will be obtained in other less classical computing architectures.**

Keywords — **Topology, Component-Labeling, Adjacency Tree, Image Processing, Parallelism.**

I. INTRODUCTION

The starting point in our work is that topology is the ideal scenario for promoting parallelism, although it drives to less classical approaches. The nature of the topological properties goes necessarily from local-to-global relations. The power of our method resides in that topological magnitudes are, by definition, robust under deformations, translations and rotations.

Up to now, the only topological invariant that has been calculated using a fully parallel computation is the Euler number [3]. Other authors have recently proposed other parallel algorithms that compute some aspects of the homological properties of binary images [13]. In [4], a digital framework for parallel topological computation of 2D binary digital images based on a sub-pixel scenario was developed, modeling the image as a special abstract cell complex [11], in order to facilitate the generalization of this work to images of higher dimensions. Still, topological approaches in that sense are rare in the literature.

Within a purely discrete level, combinatorial versions of CW-complexes, called abstract cell complexes (ACC, for short [11]), can be used for a correct algorithmic development. They are formed of basic elements (representing the cells using topological coordinates) of different dimensions together with a bounding function describing the combinatorial relationship "to be in the boundary of". Different definitions of ACCs can be found in the literature (see [29] for a thorough survey).

To sum up, we construct our scaffolding on the basis of the two following basic topological properties: "being adjacent to" and "being surrounded by". Moreover, we take advantage of the powerful duality properties that the topological invariants of connected components and holes have in the context of 2D binary digital images based on square pixel. In other words, we exploit the duality that the holes of 4-adjacent CCs (connected components) must be 8-adjacent CCs and vice versa (see Fig. 1). Finally, our algorithms use only trees as their basis. Each CC is then described by only one tree, which is connected to another CC tree by only one edge. Our framework allows us to extend the parallelism to every single pixel, in such a way that all of them do the same operations without any real dependence among them.

When writing the code, we must carefully estimate the number of operations, the memory consumption, and, the most important aspect, the ratio of memory accesses per pixel, if a fast execution is required. In fact, this last parameter is in many occasions a measure of the final algorithm performance [23].

In this paper we summarize how a pure topological framework can extend because the degree of parallelism to every single pixel. These novel image processing methods can sensibly decrease computation times if enough PEs (Processing Elements) were available.

II. RELATED WORKS

In relation to the representation of digital objects or, alternatively, binary digital images, various topological models have been exhaustively used. Adjacency trees (also called topological, inclusion or homotopy trees [2, 16, 17], and here AdjT, for short) offer a classical region-based representation in terms of rooted tree of

[1] Department of Computer Architecture and Technology. University of Seville. Spain.
[2] Department of Applied Mathematics. University of Seville. Spain.
Corresponding author: fdiaz@us.es

certain topological and spatial properties of the connected components in a binary image. Within an AdjT, each node represents a distinct foreground (FG) or background (BG) component, and an edge between two nodes means that one of them is surrounded by the other. The root in an AdjT always represents the unique BG component “surrounding" the image (if it does not exist, it can be artificially created) and two 2D binary digital images are topologically equivalent if and only if their AdjTs are equivalent. An example of an AdjT of the binary image in Fig. 1 (Left) is shown in Fig. 1 (Right).

Aside from image understanding [18] and mathematical morphology applications [7, 10, 15], AdjTs have encountered exploitation niches in geoinformatics, dermatoscopics image, biometrics, etc. (see [3, 5, 6] for instance). Therefore, finding fast algorithms for segmenting and computing the AdjT of a 2D digital binary image is crucial for solving important problems related to topological interrogations in the current technological context. It is evident that the compression of those nodes of a CCL tree (CCLT) satisfying the neighboring condition “having the same color", directly yields to the AdjT.

Connected component labeling (CCL) of binary images is one of the fundamental operations in real time applications, like fiducial recognition [6] or classifying objects as connected components (CCs). The labeling operation transforms a binary image into a symbolic matrix in which every element (pixel) belonging to a connected component is assigned to a unique label. Currently, there are mainly four classes of CCL algorithms: Multi-scan algorithms, Two-scan algorithms, Tracing-type algorithms and Hybrid algorithms mixing the previous ones. All of them (including the fastest one) use raster or tracing-type approaches, scanning the whole binary image or its contours in a sequential manner. For instance, they can label the first pixel; and then, the second one is labelled as a function of the first pixel label. This local processing runs progressively until the last pixel is reached. This fact necessarily implies real data dependencies between the labeling of one pixel and the previous one, which restricts from using a pure parallel approach. In terms of time complexity, this means that linear order *O(N)* (being *N* the number of pixels) cannot decrease independently of the number of available processing units.

Implementations for computing topological magnitudes can be achieved using classical approaches. These algorithms would contain two main stages: *1)* the scanning phase where provisional labels are sequentially assigned to pixels depending on their neighbors, *2)* and some kind of union-find technique [33] to detect and process label equivalence information of the previous assignment. Still, there is some space for parallelism when codifying scan or tracing-based CCL algorithms.

For example, dividing the image into strips is a classical data partition technique for obtaining parallelism. The second stage must then use a more sophisticated union-find technique for the provisional labels to get to the CCL. Using this classical *divide-and-conquer* approach, many works have addressed different implementations [8, 10, 15] including tuning parallel algorithms for specific computers [1]. The issue is that this division necessarily implies more data dependences between the strips in which the original image was divided (it makes harder the union-find stage). Thus, a pure parallel approach is not allowed.

Other interesting topological representations of digital images are appearing in the last years, thus leading to successful applications, for instance, in the field of image registration and matching. Most of them are hierarchical representations, which can be categorized into two classes: inclusion trees and partition trees. Leaves in inclusion trees are often image extrema, and inner nodes are formed by region growing from the leaves until the root which covers the whole image. In general, any cut of an inclusion tree does not form a complete partition of the underlying image. Typical examples are Max- and Min-tree and Tree of Shapes, which combines both of them [27]. Partitioning trees, on the other side, are initialized from an image partition. Then they rely on iterative merges of small regions at finer scale into larger regions at coarser levels. One of the most commonly used are Binary Partition Trees (BPT), α-trees and ω-trees [28]. More concretely, the α-tree, was first introduced to avoid relying on an ordering relation among image pixels (as in Max- and Min-trees). It is based on representing quasi-constant color regions of the original image.

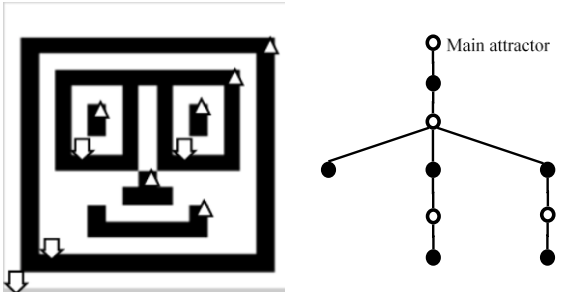


Fig. 1. Left: possible black attractors (little triangles) and white attractors (downwards arrows) for a face-like image. Holes of 4-adjacent CCs are 8-adjacent CCs and vice versa for 2D binary digital images. Right: AdjT (Adjacency tree) of the image. The main attractor is the representative of the white component that surround the whole image.

III. Topological Approach

Topological analysis of digital images studies their degree of connectivity, defining specific adjacency relations between pixels as “local neighborhood measures”. Thus, connectivity and adjacency are the key concepts in topological methods. Correctness of a framework prevents from paradoxes when an image representation is pursued. In relation to topological frameworks (Homological Spanning Forest, HSF in our case), we take advantage of the powerful duality and isotopic properties that the topological invariants of connected components and holes have in the context of 2D binary digital images based on square pixels. In fact, our method starts from an AdjT at pixel’s level and computes an AdjT at CC’s level. Let us develop this notion with a simple example.

When an object is discretized into a 2D image, it is obvious that all pixels can be linked as a tree, simply by connecting adjacent pixels using some trivial criterion for all of them (Fig. 2, (1)). For instance, the edge goes

to the South, if not possible to the West (we call it a South-West or simply SW-criterion). Note that this tree can be built independently of pixel colors. To sum up, we can state that the whole 2D image can be represented by one root pixel, which we call "attractor" (because it will attract the rest of pixels when building the tree in our framework).

Let us introduce two objects in the image, having different colors to distinguish them. According to the previous consideration, each object must be represented by only one tree, which means that one of the objects must contain the attractor of the whole image, and the other object can be represented by an attractor that falls into the previous object (Fig. 2, (2)). Conditions for one pixel to be considered being an attractor depends on adjacency criteria. Most common criteria in labelling algorithms are 8-adjacency for black pixels and 4-adjacency for white ones. In this case, tree edges connecting two pixels of different colors are candidates to attractors (Fig. 3).

Still, guessing the correct directions so that each object can be embodied by a tree cannot be achieved by local criterions. Instead, we need a global knowledge of the objects to find the correct direction for every pixel; this is patent for a spiral-like object. For instance, if we used a NE criterion for dark and SW for white pixels, cycles can appear. This is the case of Fig. 2, (3), where a simple black 'L' shape produces an undesirable situation: both black and white objects have two attractors. Thus a cycle comes out.

Thus, a meticulous strategy that allows a fast detection of incorrect directions and their parallel corrections must be found. For two dimensional digital images this can be achieved in two stages: 1) using a NE (North-East) criterion for dark pixels and a SW one for the white pixels; 2) Once a cycle is detected, transport two edges so that we get to the correct HSF (having one tree per object) (Fig. 2, (4)). Dashed arrows are the new transported edges.

To sum up, edges that connect different colors are candidates (attractors) of frontiers between CCs. False attractors (in case a cycle is detected) can be transported to get to the correct HSF. In the end, any tree covering the image plus the region frontier candidates is an instance of a connectivity tree that holds the complete information of the image. Further details about how to implement a fully parallel implementation of this process are detailed in next sections.

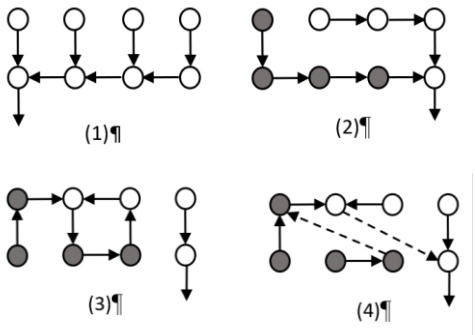


Fig. 2. (1) Any 2D image can be connected as a single tree. (2) Guessing the correct directions so that each object is a tree (3) Using NE criterion for dark and SW for white pixels can produce cycles. (4) A transport pair to get the correct HSF (one tree per object).

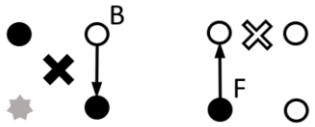


Fig. 3. The two unique possible patterns for attractors. Left: white attractor that is connected to an 8-adj black set of pixels. Right: black attractor connected to a 4-adj white set of pixels. Grey star represents a pixel of any color

IV. PARALLEL PROCESSING KEYS

For the sake of clarity, from now on, this paper concentrates on the parallel procedure to label a B/W image of mxn pixels (the problem known as CCL, Connected Component Labelling). Current CCL solutions are fully sequential on their first stage. That is, the provisional label of a pixel is written as a function of some set of the previous one (Fig. 4). In fact the strength of fastest CCL algorithm (according to YACCLAB [22]) resides on the use of a big window of neighboring pixels and a very ingenious way to reduce the hundreds of combinations of this window into a few dozens of cases (a Decision Tree or Table strategy) [24].

After the sequential stage, a 'Union-Find' phase combines and relabel those labels that are detected to belong to the same CC. This will be the case of labels 1 and 3 in Fig. 4; their label equivalence would be discovered when approaching the most South-East black corner.

1	1	2	2	2	2	3
1	1	1	2	2	2	3
4	4	1	?			

Fig. 4. A B/W image showing a sequential labelling (using a South-East direction). When a new CC is found, it is assigned a new incrementing label. Next pixel (e.g. that marked with '?') must be labelled as a function of the previously labelled pixels.

Our previous topological framework allows to build in a fully parallel manner all the labelling. Instead of assigning a non-meaningful label to each pixel, we can set them with the jump distance to their attractor. Thus, true attractors are to be set as 0, whereas false attractor (after a transport) will be given a jump distance to the corresponding attractor (see Fig. 5). There are two main phases to proceed with the jump distance computation.

+1	0	-1	+4
-4	+1	0	0

+1	+6	-1	+4
-4	+1	-2	0

Fig. 5. Left: Jump distances of Fig. 2, (3). A linear address distance is followed, being the jump of +/-4 a change of row (because image has 4 columns). Right: New jump distances assigned to false attractors after the transport is done.

A. *First stage: from local to global jump distance.*

In the first stage, every pixel (in parallel) computes its jump distance to its attractor. This can be done by using exponentially growing jump distances in a logarithm number of iterations. Fig. 6 shows an example for 9 adjacent pixels, which can be completed after three iterations. Arrows expresses the memory accesses that every pixel must do. For each reading, each pixel must

add its previous distance with the new read value. As far as we know, the first work that proposed a similar scheme was [25] with the purpose of producing highly efficient Monte Carlo simulations for two and three-dimensional critical Ising models.

Similar procedures can be extended for any dimension. Further details for applying this phase to 2D binary images can be found in our previous paper [26], which it is shown that this phase can be executed in *log2(m+n-1)* iterations at most, due to its exponential nature. Supposing that we have *p* PEs, their complexity is *O((mn/p)log(m+n))*, thus being this phase usually the most time consuming.

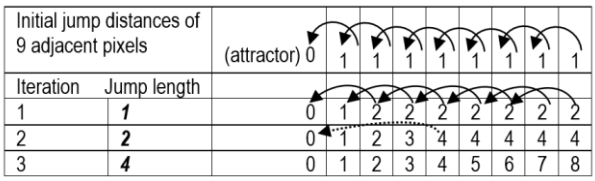


Fig. 6. Three iterations of parallel jump distance computation for 9 adjacent pixels. Most left pixel is the attractor. Arrows expresses the memory accesses that every pixel must do (only one pointed arrow is depicted for the second iteration for clarity purposes).

B. *Second stage: transports.*

The second stage consists of the parallel transports of pairs of false black and white attractors. This supposes a transport of edges until a unique tree existed, and no cycle remains. Any transport implies the updating of two jump matrix elements (or equivalently, redirecting two edges for each pair of false attractors). If black and white pixels conformed tree structures and followed different directions when doing previous jump distance computation, all transports can be done in parallel if the next *concurrency condition* is detected for each possible transport. This perfect concurrency of executing many transports is guaranteed since for each transport there are two travels through trees up to their corresponding roots (attractors). The condition that the beginning pixel must be the same as the destination after the two tree travels ensures the unicity of the pair to be cancelled (Fig. 7). Hence transport phase has no need for any critical section or atomic clause. This cannot be ensured in classical Union-Find techniques.

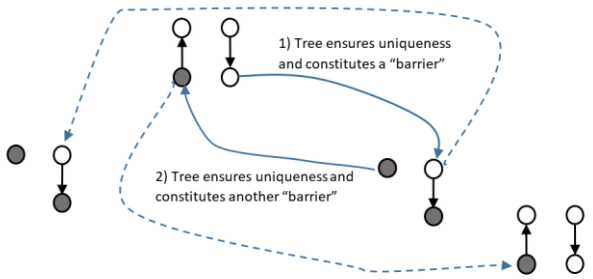


Fig. 7. Condition for guarantying the perfect concurrency of executing many transports in parallel. First, from the above black attractor a tree ensures finding a unique white attractor, which additionally constitutes a "barrier" of white pixels. Going back (step 2) from the black adjacent pixel to the white attractor also ensures unicity. Finally, dashed arrows represent the new jump distances to be computed after the transport is done.

This stage must execute several pairs of cycle searching. Although this phase seems to be tricky, if there were more PE than attractors, its timing complexity is reduced to a few iterations. In [26] it is found that the number of iterations reached a maximum of six pairs even for the most problematic images (big random images -16 Mpixels- having a 50% of black pixels). Conversely, it was only one for the real images tested (having a size until 2 Mpixels). The worst-case scenario of this phase is left for future research.

At the end of these two stages, we get to a new representation of the 2D image, in which any matrix element contains a jump distance to its true attractor, that is, to the root of the tree that represents the whole CC (see Fig. 10). Obtaining the AdjT is quite straightforward; simply by looking for each attractor the jump distance of its adjacent opposite color pixel (which goes to a new attractor).

Jump distance information is the basis for many other topological representations; some of them are shortly discussed in section VI.

V. Experimental Results

Two complete implementations were done in C++/OpenMP. The first was a direct translation of a previous MATLAB/OCTAVE implementation presented in [26]. The second is a more optimized version, whose results have been submitted to the journal 'Pattern Recognition Letters'. The server where tests were carried out was an Intel Xeon E5 2650 v2 with: 2.6 GHz, 8 cores, 8x32 KB data caches, Level 2 cache size 8x256 KB, Level 3 cache size 20 MB, maximum RAM bandwidth: 59.7 GB/s. Experiments were run 25 times and mean times were collected.

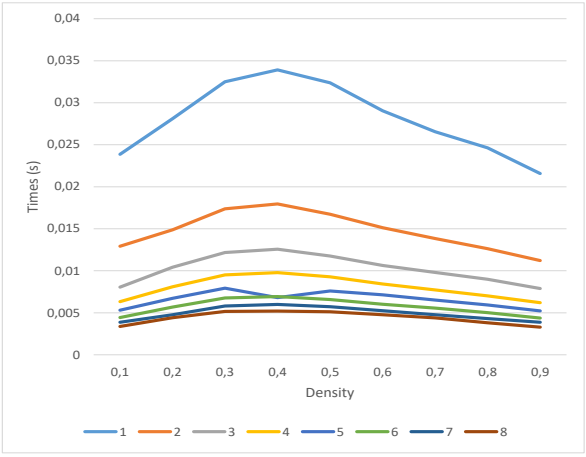


Fig. 8. Times for 1 to 8 threads as a function of density (random images).

For the first implementation, when optimization is poor, the scalability is very high. Thus, speedup (time for various threads divided by time for 1 thread) is near the number of threads (Table I), which points out that achieved scalability is excellent for all image sizes and densities. Fig. 8 depicts times for a set of 512x512 pixel images with different densities, showing that processing times are very near to that of current fastest algorithm [22]. Taking into account the good scalability, we expect that our implementation ran even faster in a massive

multicore processor. Although scalability is a little inferior for real images (Fig. 9) than for random images, times are much smaller. In fact, the processing time for the only random image ("633.png", 4196 Kpixel) in this figure is even bigger than that of a real image with a double size. This is due to the higher amount of CC that random images usually have (in relation to the real ones).

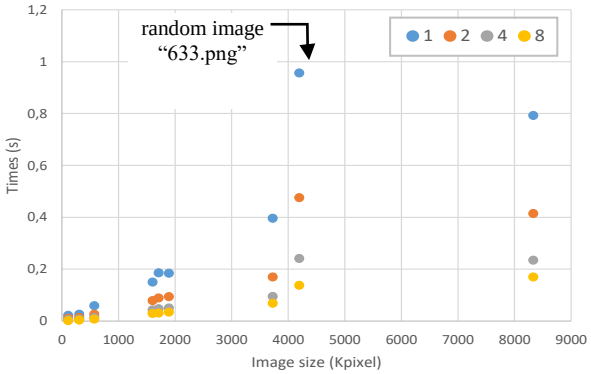


Fig. 9. Times for 1, 2, 4, 8 threads for real images.

TABLE I

SPEEDUP FOR RANDOM IMAGES OF DIFFERENT SIZES (DENSITY = 0.9).

#threads	256x256	512x512	1024x1024	2048x2048
2	1,88	1,92	1,94	1,95
3	2,65	2,73	2,89	2,82
4	3,39	3,48	3,79	3,61
5	3,92	4,13	4,57	4,33
6	4,45	4,95	5,30	4,98
7	5,56	5,59	5,94	5,55
8	5,93	6,58	6,48	6,19

TABLE II

MEAN TIMES FOR RANDOM IMAGES OF DIFFERENT SIZES AND DENSITIES. BBDT IS FROM [22][23] AND HSF IS OUR METHOD (#THREADS IN BRACKETS).

Size	BBDT	HSF (1)	HSF (2)	HSF (3)	HSF (4)
1024	0,009	0,023	0,100	0,095	0,127
4096	0,029	0,064	0,144	0,130	0,158
16384	0,102	0,221	0,296	0,243	0,268
65536	0,374	0,839	0,896	0,624	0,666
262144	1,444	3,305	3,040	2,229	2,129
1048576	5,706	13,239	10,888	7,660	7,229
4194304	23,662	65,644	42,131	32,372	28,553
16777216	117,962	338,320	210,366	154,011	129,632

Size	BBDT	HSF (5)	HSF (6)	HSF (7)	HSF (8)
1024	0,009	0,128	0,106	0,110	0,114
4096	0,029	0,149	0,129	0,135	0,127
16384	0,102	0,235	0,199	0,203	0,189
65536	0,374	0,546	0,452	0,410	0,397
262144	1,444	1,885	1,391	1,252	1,180
1048576	5,706	6,384	4,811	4,305	3,836
4194304	23,662	25,878	20,275	18,742	18,170
16777216	117,962	110,767	90,458	80,713	78,685

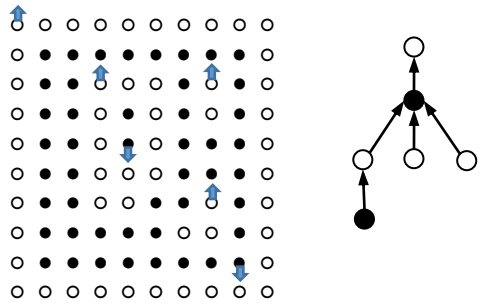
Besides, Table II shows the results from the second (optimized) version of our method compared with the BBDT method, which is the currently fastest CCL algorithm according to [22]. The optimization of our code introduces more than 7x speedup with respect to the timing of Fig. 8, but decreases the multithread speed-up to only 4x for 8 threads. Of course, for little images the extra overhead time (introduced by OpenMP when creating the threads) hinders speedup. This supposes that speedups are also decreased for medium images.

For this second optimized implementation, we can beat the fastest sequential CCL algorithm when executing on a convenient number of cores (in general, 5 or 6 threads in our experiments with medium/big random images, see Table II). However, scalability begins to be less high because data accesses come to be a bottleneck.

Finally, an additional advantage of our approach is that it presents lower deviation for a same size and different densities than the BBDT method. This is manifest when processing images of very different textures.

VI. FUTURE WORK: OTHER TOPOLOGICAL REPRESENTATIONS

Jump distances define another image representation that allows to obtain topological measures straightforwardly (Fig. 10).



0	-1	-2	-3	-4	-5	-6	-7	-8	-9
-10	77	76	75	74	73	72	71	70	-19
-20	67	66	0	-1	-2	62	0	60	-29
-30	57	56	-10	10	-12	52	-10	50	-39
-40	47	46	-20	0	-22	42	41	40	-49
-50	37	36	-30	-31	-32	32	31	30	-59
-60	27	26	-40	-41	1	22	9	20	-69
-70	17	16	15	14	13	0	-1	10	-79
-80	7	6	5	4	3	2	1	0	-89
-90	-91	-92	-93	-94	-95	-96	-97	-98	-99

Fig. 10. Up Left: A 10x10 B/W image Attractors are marked with upwards (for the white) downwards (for the black) arrows. Up Right: Its AdjT. Bottom: The Jump distance matrix. Black pixels follow a SE criterion (positive values in general) and white ones a NW (negatives values in general). Attractors are assigned a value 0 (highlighted with shadow).

Going further, digital images of any dimension and with multiple object inside (that is, color images) requires more powerful topological description. In this case, we can benefit from other duality topological properties, like object/border. This concept needs to declare a convenient abstract cell complexes (ACC, [11]) for dealing with color images. Exploration of this approach demonstrates that two dimensional color images can be treated with 4 cells per elemental PE [4], using cells of dimensions 0, 1 and 2. An example of a color image and an elemental PE is found in Fig. 11. Each PE covers a pixel in a digital image and can hold the information related to flat color zones and their borders (called cracks in [4]), that is, a region-contour

HSF [31]. That is, a complete image can be composed as many trees as correlative dimensions (*0-1* tree and *1-2* tree for the case of Fig. 11). Region-contour information can be seen in Fig. 11 (Right) as a set of 0, 1, 2 cells for flat zones (regions) and a selected set of 1, 2 cells drawing the region interfaces (contours as black segments).

For doing so, we must proceed in a similar manner to that explained previous sections, that is, cells (having their topological coordinates) of different dimensions must be joined together with a bounding function, and a combinatorial optimization process must compress the whole image into just one pixel. The relationship "to be in the boundary of" for the different regions must be efficiently computed and stored to preserve the topological information of the image. Each of these relations can be seen as a division of one tree into several sub-trees (Fig. 11, Right).

Besides, using this previous topological information, a potential idea consists of introducing color order relations among sub-trees to extract more sophisticated features. In this sense, during the last decade, features based of pure topological relations (Max- and Min-tree, Tree of Shapes, Binary Partition Trees, α-trees and ω-trees, etc.). Recently the so-called Tree-Based Morse Regions (TBMR, [32]) determines local invariant "interest" points, with the same complexity as classical MSER (Maximally Stable Extremal Regions), and a repeatability on par with state-of-the-art methods. In addition, it obtains a significantly higher number of features, being both accurate and robust enough to be applied to image registration and 3D reconstruction.

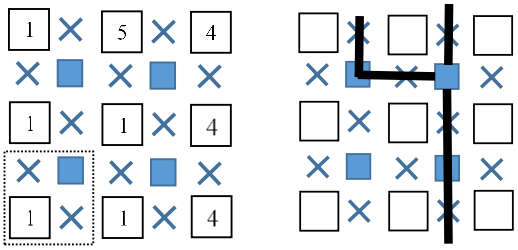


Fig. 11. Left: a fragment (9 pixels) of a color image. Numbers represent color values of the original image pixels. For the ACC representation, numbers are 0-cells, crosses are 1-cells and solid squares are 2-cells. At the most bottom left corner, the dotted square is an elemental PE composed of 4 cells. Right: a possible contour tree (divided into subtrees) containing the border information of the image represented by 1 and 2 cells.

For two dimensional objects, only two homology groups must be considered: those representing connected components and holes. However, this topological framework can be extended to high dimensional images by defining the proper elemental PE that allows a complete topological representation and, afterwards, by building the different *k-(k+1)* trees (being *k* a dimension) in the most effective way [30]. Then, objects immersed on the *nD*-image would be represented by homology groups of dimensions *0, 1, …, n-1*.

CONCLUSIONS

Most of the image processing algorithms part from tracing-type or scan/raster methods. This fact necessarily introduces data dependences between the processing of one pixel and the previous one, which prevents pure parallel implementations. We describe a very different approach based on a pure topological framework, which allows to implement fully parallel algorithms. This yields to an image representation that avoids complex data structures. In fact, all the processing can be done using matrixes (with the same indexation as the original image) and element-wise operations. Theoretical time complexity orders of our topological approach for an image of $m \times n$ pixels is near $O(\log(m+n))$. Being a consistent topological framework, this method can be extended to color *n*-dimensional images.

ACKNOWLEDGEMENT

This work has been supported by the Spanish research projects MTM2016-81030-P (AEI/FEDER,UE)) and TEC2012-37868-C04-02 of Ministerio de Economía y Competitividad and the VPPI of the University of Seville.

REFERENCES

[1] P. Bhattacharya. Connected component labeling for binary images on a reconfigurable mesh architecture. Journal of Systems Architecture, 42(4):309-313, 1996.
[2] O.P. Buneman. A Grammar for the Topological Analysis of Plane Figures. Edinburgh Univ. Press, pp. 383-393, 1969.
[3] F. Chiavetta, V. Di Ges. Parallel computation of the Euler number via connectivity graph. Pattern Recognition Letters 14, 849-859, 1993.
[4] F. Diaz-del-Rio, P. Real, D., Onchis: A parallel Homological Spanning Forest framework for 2D topological image analysis. Pattern Recognition Letters 83, 49-58, 2016.
[5] A. Cohn, B. Bennett, J. Gooday, N. Gotts: Qualitative spacial representation and reasoning with the region connection calculus. GeoInformatica 1(3), 275-316, 1997.
[6] E. Costanza, J. Robinson. A region adjacency tree approach to the detection and design of fiducials. Video, Vision and Graphics, pp. 63-99, 2003.
[7] R. Cucchiara, C. Grana, A. Prati, S. Seidenari, G. Pellacani. Building the topological tree by recursive FCM color clustering. 16th IEEE ICPR, 1, pp. 759-762, 2002.
[8] S. Gupta, D. Palsetia, M.M.A.Patwary, A. Agrawal, A.N. Choudhary. A new parallel algorithm for two-pass connected component labeling, in: IEEE IPDP Symposium, pp. 1355-1362, 2014.
[9] H. J. Heijmans. Connected morphological operators for binary images, Comput. Vis. Imag. Understand., 73 (1), pp. 99-120, 1999.
[10] O. Kalentev, A. Rai, S. Kemnitz, R. Schneider. Connected component labeling on a 2d grid using CUDA. J. Parallel Distrib. Comput. 71, 615-620, 2011.
[11] V. Kovalevsky.: Algorithms in Digital Geometry Based on Cellular Topology. 10th IWCIA, Springer Berlin Heidelberg, vol. 3322, pp. 366-393, 2004.
[12] R. Keshet.: Shape-tree semilattice. J. Math. Imag. Vis., 22 (2-3), pp. 309-331, 2005.
[13] A., Murty, V., Natarajan, S., Vadhiyar.: Efficient homology computations on multicore and manycore systems, in: 20th Annual International Conference on High Performance Computing, pp. 333-342, 2013.
[14] NVIDIA, Cuda C best practices guide version. http://developer. nvidia.com/. Oxley, J.G., Matroid theory. volume 3. Oxford University Press, 2017.
[15] M. Patwary, M. Ali, P. Refsnes, and F. Manne. Multi-core spanning forest algorithms using the disjoint-set data structure. In 26th IEEE IPDP Symposium, pp. 827-835, 2012.
[16] REDHOM, Redhom. http://redhom.ii.uj.edu.pl/, Institute of Computer Science, Jagiellonian University, 2017.
[17] V. Ranwez, P. Soille, Order independent homotopic thinning for binary and grey tone anchored skeletons, Pattern Recognition Letters 23 (6), 687-702, 2002.
[18] A. Rosenfeld. Adjacency in digital pictures. Inf. Control 26, 24-33, 1974.
[19] J. Serra. Image Analysis and Mathematical Morphology. Academic Press, 1982.

[20] J. Stell and M. Worboys.: Relations between adjacency trees. Theo. Comp. Sci., 412 (34), pp. 4452-4468, 2011.
[21] S. Williams, A. Waterman, D.A. Patterson. Roofline: an insightful visual performance model for multicore architectures. Commun. ACM 52, pp. 65-76, 2009.
[22] YACCLAB - Yet Another Connected Components Labeling Benchmark. https://github.com/prittt/YACCLAB, 2017.
[23] Grana, C., Bolelli, F., Baraldi, L., Vezzani, R., 2016. YACCLAB - Yet Another Connected Components Labeling Benchmark, in: 23rd International Conference on Pattern Recognition, ICPR.
[24] Grana, C., Borghesani, D., Cucchiara, R., 2010. Optimized block-based connected components labeling with decision trees. IEEE Transactions on Image Processing 19, 1596–1609
[25] Swendsen, R.H., Wang, J., 1987. Non-universal critical dynamics in Monte Carlo simulations. Phys. Rev. Lett. 58, 86–88.
[26] Díaz-del-Río, F., H., Molina-Abril, P. Real, 2019. Computing the component-labeling and the adjacency tree of a binary digital image in near logarithmic-time. Computational Topology in Image Context, Lecture Notes in Computer Science, Springer 11382, 82–95.
[27] Soille, P. , Constrained Connectivity for Hierarchical Image Partitioning and Simplification, IEEE Transactions on Pattern Analysis and Machine Intelligence, V 30, N. 7, pp 1132–1145, 2008.
[28] Petra Bosilj, Ewa Kijak, Sébastien Lefèvre. Partition and Inclusion Hierarchies of Images: A Comprehensive Survey. Journal of Imaging, MDPI, 2018, 4 (2), pp.1-31.
[29] R. Klette , Cell complexes through time, in: Proceedings of SPIE 4117, vol. 4117, 20, pp. 134–145 .
[30] Real, P., Diaz-del-Rio, F., Onchis, D.: Toward Parallel Computation of Dense Homotopy Skeletons for nD Digital Objects. In International Workshop on Combinatorial Image Analysis, pp. 142-155. Springer, 2017.
[31] Real, P., Diaz-del-Rio F., Onchis, M. Labeling color 2D digital images in theoretical near logarithmic time. 17th international Conference on Computer Analysis of Images and Patterns. Istad, Suecia. 2017
[32] Y. Xu, P. Monasse, T. Géraud, L.Najman. Tree-Based Morse Regions: A Topological Approach to Local Feature Detection October 2014IEEE Transactions on Image Processing 23(12).
[33] Lifeng He, Xiwei Ren, Qihang Gao, Xiao Zhao, Bin Yao, Yuyan Chao. The connected-component labeling problem: A review of state-of-the-art algorithms. Pattern Recognition 70 (2017) 25–43

Emulador HEVC INTRA en Matlab

Javier Ruiz Atencia, Otoniel López Granado, Manuel Pérez Malumbres, Miguel O. Martínez-Rach[1]

Resumen— **Se ha diseñado un emulador HEVC Intra en Matlab con el objetivo de proporcionar a la comunidad científica y educativa una herramienta que implementa las diferentes etapas de codificación Intra del estándar HEVC y obtiene resultados mediante curvas rate-distortion y valores BD-Rate para distintas métricas objetivas de calidad. Las etapas de transformación, cuantización uniforme y perceptual, predicción Intra y reconstrucción han sido validadas con el software de referencia HM 14.0. Se utiliza un particionado en CUs de tamaño fijo. El desarrollo en Matlab permite probar modificaciones de diseño del HEVC relacionadas con estas etapas de forma más sencilla que utilizando el software de referencia. Se detalla su funcionalidad y se muestran resultados de la salida obtenida al codificar diferentes secuencias de vídeo.**

Palabras clave— **HEVC, emulador, transformación, cuantización, predicción Intra, Matlab**

I. Introducción

EN el proceso de investigación sobre posibles mejoras de los codificadores de video o imagen, los investigadores suelen trabajar con el software de referencia, para que finalmente el producto de su investigación sea compatible con el estándar para en el que se enfocan. La mayoría de los estándares recientes de codificación de video, H.264 o HEVC por ejemplo, se centran en definir las características que tiene que tener el bitstream para poder ser leído por un software compatible por el estándar.

De esta manera mucha de la investigación relacionada con mejoras en el HEVC se centra en poder modificar, mejorar u optimizar el codificador para que su resultado genere un bitstream compatible pero con mejoras en calidad, rendimiento, coste computacional, etc.

Como hemos dicho los investigadores suelen utilizar el software de referencia del codificador para incluir o probar sus ideas o mejoras. Trabajar sobre este software de referencia es a menudo costoso, puesto que la curva de aprendizaje del mismo y la complejidad del mismo son importantes.

Son pocos los casos en los que el investigador plantea una mejora en alguna de las etapas de codificación/decodificación teniendo la certeza de obtener los resultados esperados tras insertarlo en el estándar. Lo habitual es que se tenga que recurrir a distintas modificaciones iterando varias veces en un bucle de refinamiento de su teoría al observar los resultados obtenidos.

No olvidemos que las imágenes y videos se tratan como matrices de datos sobre los que el software opera para transformarlos, filtrarlos, reducirlos, codificarlos etc. y esta naturaleza matricial de la imagen y el video es ocultada por el lenguaje de programación utilizado en el software de referencia, habitualmente C++, con el uso de punteros y complejas y no siempre bien documentadas, estructuras de datos u objetos. Cuando la mejora que plantea el investigador se basa por ejemplo en procesar la matriz de datos para aplicar nuevos filtros o realizar operaciones entre distintas matrices es mucho más intuitivo aplicar un lenguaje de programación basado en matrices como Matlab.

Resultaría más práctico para el investigador que el bucle de codificación de la idea, ejecución, análisis de resultados, estudio y recodificación de la misma se haga directamente sobre matrices, por ejemplo cuando en sus propuestas se vean involucradas etapas donde la información es intrínsecamente matricial como las de transformación, predicción, cuantización y reconstrucción del HEVC entre otras.

Sin embargo no todas las líneas de investigación relativas al HEVC se beneficiarían de un HEVC en Matlab, por ejemplo aquellas cuyo objetivo fuese la mejora del coste computacional. Además hay que tener en cuenta que Matlab consume muchos recursos de la máquina y la velocidad de ejecución no es comparable a la obtenida por el software compilado en C++.

Pero para aquellos casos como los mencionados y aunque finalmente las propuestas deban ser implementadas en el software de referencia, una herramienta compatible con la ejecución del software de referencia del HEVC implementada en Matlab sería muy interesante. Cuando hablamos aquí de compatibilidad queremos decir que los resultados de cada una de las etapas implementadas son 100 % coincidentes con el resultado (bit a bit y valores enteros o flotantes) con aquellas etapas del software de referencia ante la misma entrada y la misma configuración, es decir, aplicando los mismos procesos. De esta forma se podría asegurar que los resultados obtenidos en Matlab por ejemplo en una variación que mejore de la etapa de predicción Intra, serían los mismos si se implementara esa mejora en el software de referencia. Con la ventaja de haber trabajado sin la interferencia mencionada del lenguaje de programación.

En la búsqueda de una herramienta con las características de la presentada hemos encontrado varias herramientas que proporcionan un análisis completo del bitstream del HEVC con muchas posibilidades y opciones de visualización, algunas de las cuales también de open-source GPL [1]. También existen distintas librerías, módulos e implementaciones del codificador HEVC con código open-source y distintas al software de referencia pero en C++ ninguna

[1]Dpto. de Ingeniería de Computadores, Universidad Miguel Hernández de Elche, e-mail: mmrach@umh.es.

en Matlab. Algunos autores publican el código en Matlab utilizado en su investigación para poder ser reproducido y ampliado [2]. En esta línea en [3] los autores presentan una clase Matlab que utilizando MEX (integración C++ en Matlab) dan acceso al código del motor CABAC de la HM. Esta herramienta es muy interesante para aplicar a nuestro proyecto en futuras versiones. Únicamente hemos encontrado como trabajos MastherThesis implementaciones de la predicción Intra en Matlab [4] y [5] proporcionando ésta última una herramienta que incluye la transformada y la predicción Intra en Matlab bastante cercana a lo que pretendíamos pero no implementa la cuantización y no genera exactamente los mismos resultados que el software de referencia en las etapas implementadas, con lo que la convierten en una herramienta muy interesante para conocer estas etapas desde un punto de vista docente pero no garantizan la compatibilidad que buscamos.

Por tanto en este trabajo hemos desarrollado una herramienta en Matlab que actualmente es ejecutable únicamente desde la línea de comandos para emular al codificador HEVC con las características mencionadas en la sección II. El resto de las secciones se estructuran de la siguiente manera. En el capítulo III se detalla cada uno de los módulos que compone el emulador. Junto a cada módulo se adjunta su diagrama de flujo para facilitar el entendimiento del mismo. En el capítulo IV se exponen a modo de ejemplo resultados obtenidos con esta herramienta. Finalmente, en el capítulo V se da una visión global y resumida del artículo y se definen las líneas futuras de investigación.

II. Características del emulador

El diseño y desarrollo del emulador HEVC en Matlab se ha realizado siguiendo las directrices indicadas en [6] y usando la documentación y el propio código del software de referencia HM [7]. A continuación se muestran los bloques HEVC codificados en Matlab y cuyos resultados están validados con el software de referencia:

- Transformación y cuantización (así como sus inversas) para tamaños de bloque desde 4×4 hasta 32×32.
- Cuantización perceptual (Weighting Matrix) para todos los tamaños del estándar.
- Obtención de las muestras de referencia para las predicciones y sus filtros cuando corresponden.
- Cálculo de las 35 predicciones Intra (modos Planar, DC y 33 modos angulares).
- Obtención del residuo y reconstrucción de imagen.

El software se completa con la implementación de las siguientes características:

- Propuesta de matriz de cuantización perceptual (y su inversa) para tamaños de bloque 4×4 (CSF).
- Selección del mejor modo de predicción mediante:
  - mínimo SATD.
  - mejor R/D donde el Rate se obtiene vía la entropía de primer orden, no usando CABAC.
- Particionado de imágenes en bloques de tamaño fijo (4, 8, 16 y 32 píxeles).
- Procesa solo luminancia para imágenes en Intra mode o secuencias YUV 4:2:0 All Intra.
- Fichero de configuración con parámetros variables y proceso por lotes.
- Obtención de gráficos R/D y estadísticos para las distintas QPs utilizando diferentes métricas de calidad.
  - PSNR, SSIM [8], MSSSIM [9], VIF [10], VIFP [10], PSNRHVS [11] y PSNRHVSM [12].
- Informe de ganancias Bjøntegaard (BD-Rate y BD-Distortion) adaptadas a las distintas métricas de calidad para cada gráfica R/D obtenida.

Con todo esto, la herramienta es capaz de codificar imágenes o vídeos con formato de origen YUV 4:2:0 (únicamente la componentes de luminancia) y 8 bits de profundidad tal y como lo hace el software de referencia HM tras aplicarle el archivo de configuración modificado deshabilitando las funcionalidades no trasladadas al emulador.

III. Esquema y desarrollo del emulador

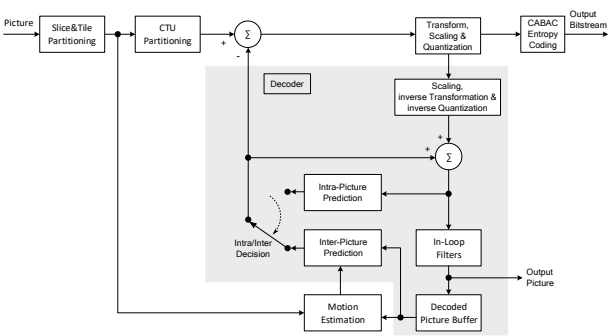


Fig. 1: Diagrama de bloques de un codificador HEVC con decodificador incorporado (*región grisácea*)

El desarrollo del emulador HEVC en Matlab se ha planificado partiendo del esquema básico del codificador HEVC (Figura 1) descartando toda la parte del estimador de movimiento (predicción Inter) así como los filtros In-loop.

La programación del código del emulador se ha realizado de forma modular, lo que permite la paralización de algunas de sus etapas.

A. EmuladorHEVC

En la Figura 2 se muestra el diagrama de flujo del archivo principal del emulador, donde se cargan los parámetros de ejecución, estos son:

- `sequences`: Variable de tipo *cell array* donde se establecen la o las secuencias a procesar.
- `dims`: Variable de tipo *cell array* donde se establecen las dimensiones en píxeles de cada una de las secuencias a procesar.

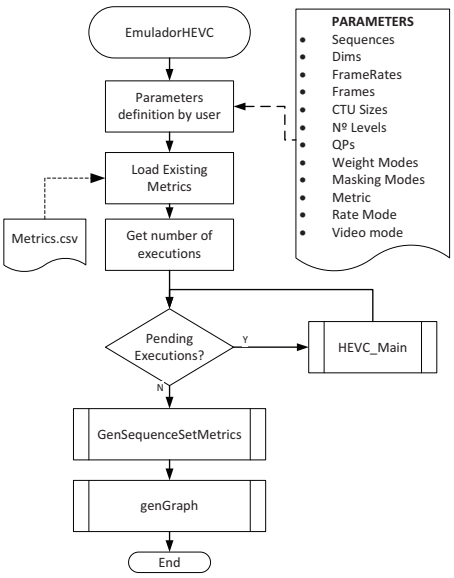


Fig. 2: Diagrama de flujo del archivo `EmuladorHEVC.m`

• `FrameRates`: Variable de tipo *cell array* donde se establece el framerate de cada secuencia.
• `Frames`: Variable de tipo *cell array* donde se especifican los frames a procesar para cada secuencia.
• `CTUSizes` y `NLevels`: Estas dos variables de tipo *double array* establecen el tamaño de CTU y su nivel de división respectivamente. De esta forma, los tamaños fijos de bloque (CU) para el particionado de la imagen vienen determinados por la expresión:

$$\text{CUSize} = \frac{\text{CTUSize}}{2^{\text{NLevel}}} \tag{1}$$

Los valores de CTU permitidos son 16 y 32, mientras que los tamaños de Niveles permitidos van desde 0 hasta 2, con lo que se pueden configurar para tamaños de bloque de 32, 16, 8 y 4.
• `Qps`: Variable de tipo *double array* permite elegir las distintas QPs para cada ejecución. Los valores permitidos corresponden con el estándar HEVC, pudiendo ser desde 0 hasta 51.
• `WeightModes`: Variable de tipo *cell array* donde se determina si se usan o no matrices de cuantización perceptual, basadas en la CSF. Las opciones permitidas son:

- `noCSF`: Utiliza las matrices de cuantización uniformes.
- `staCSF`: Utiliza las matrices de cuantización no-uniformes del estándar HEVC. Para tamaño de bloque 4×4 aplica cuantización uniforme.
- `CSF`: Utiliza las matrices de cuantización no-uniformes del estándar HEVC, excepto para tamaños de bloque 4×4, en cuyo caso se utilizan unas matrices definidas en [13].

• `Metric`: Variable de tipo *char array* donde se especifica la métrica que queremos que muestren las gráficas y para el cálculo del BD-Rate (Bjøntegaard [14]). Cuando se lanza la codificación de las secuencias se calculan todas las métricas (PSNR, SSIM, MSSSIM, VIF, VIFP, PSNRHVS, PSNRHVSM). Se reutilizan estos valores para ejecuciones con la misma configuración.
• `RateMode`: Variable de tipo *char array* donde se especifican las unidades de rate (bits, bpp, bps, kbps, Mbps).
• `videomode`: Variable *booleana* que define si los ficheros de entrada corresponden a imágenes o secuencias, en cuyo caso se promedian los valores de BD-Rate de los frames seleccionados.
• `parallelMode`: El programa permite utilizar esta variable *booleana* para la ejecución paralela en hilos de ejecución independientes.
• `print_to_pdf`: El programa permite utilizar esta variable *booleana* para generar las gráficas resultantes en formato PDF.

El programa genera un archivo `Metrics.csv`, que es la base de datos donde se guarda el resultado de las ejecuciones en formato CSV. Si por los parámetros de entrada se solicita la ejecución de alguna configuración existente en esta base de datos, se omitirá su ejecución y cargará los datos de la misma.

A partir de los parámetros de configuración y descartando los ya realizados se obtiene el número de ejecuciones a realizar. Una ejecución es una combinación única de los parámetros definidos por el usuario, por ejemplo:

```
sequences = 'BlowingBubbles_384x192_50.yuv';
dims = [384,192];
FrameRates = [50];
Frames = [1:50];
CTUSizes = [16];
NLevels = [1];
Qps = [32];
WeightModes = 'noCSF';
```

Después, para cada lanzamiento se ejecuta el proceso HEVCMain, encargado de codificar la secuencia. Una vez se han procesado todos los lanzamientos, se obtienen las métricas y se crea o se añaden nuevas líneas al archivo `Metrics.csv`. Finalmente se lanza la función `genGraph`, que muestra por pantalla las gráficas para todas las secuencias y crea los archivos `Graficas_*.csv` y `Bjontegaard_*.csv`, que contienen los puntos de las curvas y las métricas Bjøntegaard respectivamente.

B. HEVC_Main

`HEVC_Main` es proceso principal encargado de generar la codificación de las secuencias, así como su decodificación para reconstruir la imagen y guardar los archivos necesarios para posteriormente obtener las métricas de calidad. En la Figura 3 se muestra su diagrama de flujo.

El primer paso consiste en calcular el tamaño de los bloques para su particionado utilizando la Ecuación 1. La función `DefineHEVCTables`, obtiene las matrices de transformación y cuantización en función del tamaño de CU.

A continuación se extraen del vídeo original los frames a utilizar en formato YUV y BMP. Siguiendo el Esquema 3, para cada Frame de la ejecución se realizan los siguientes procesos:

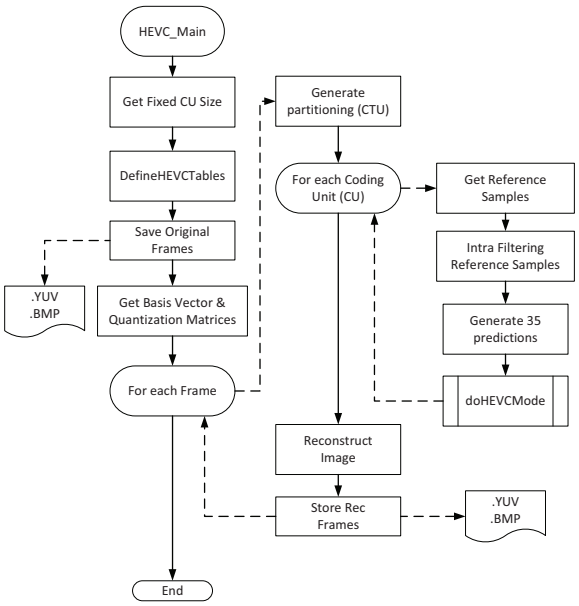


Fig. 3: Diagrama de flujo del archivo `HEVC_Main.m`

- **Particionado del frame:** Se calcula el tamaño del frame para que sea múltiplo de CU y se divide el frame en CUs para su procesamiento.
- **Generación de la matriz de recorrido:** Los CTU de una imagen se recorren en *raster scan order* y su descomposición en CUs utilizando el recorrido *Z-Scan order*(ver Figura 4), que permite tener codificados los bloques necesarios para la predicción Intra.

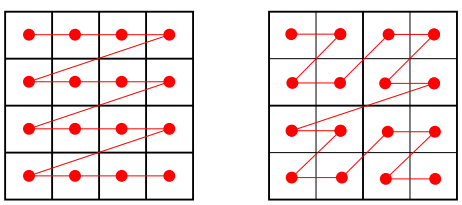


Fig. 4: Algoritmo de recorrido de bloques para un particionado de tamaño fijo. Izquierda: recorrido por CTUs (*raster order*). Derecha: recorrido por CUs (*Z-Scan order*) dentro de cuatro CTUs con un nivel de división.

- **Procesado de cada CU:** El siguiente paso consiste en procesar cada CU en el orden correspondiente, que explicaremos más adelante.
- **Guardado de imagen reconstruida:** Tras procesarse todos los CUs de un frame guardamos la imagen reconstruida en formato YUV y BMP.

El procesamiento que se realiza para cada CU es el siguiente:

- **Obtención de las muestras de referencia:** La función `GetReferenceSamples` genera las muestras de referencia de los bloques adyacentes necesarias para la predicción Intra siguiendo el algoritmo del estándar HEVC.
- **Post-filtrado de muestras de referencia:** Obtenidos los vectores de referencia, a continuación se determina si se debe aplicar el filtro de post-procesado, tal y como está descrito en [6]. En la Tabla I se muestran las diferentes opciones de filtrado según el tamaño de bloque y predicción.

TABLA I: Aplicación de filtro de post-procesado a las muestras de referencia

Tamaño de CU	**Aplicación de filtro**
4×4	No aplicar
8×8	Planar y Angular (2, 18, 34)
16×16	Todos los modos excepto DC, Angular (9, 10, 11, 25, 26, 27)
32×32	Todos los modos excepto DC, Angular (10, 26)

- **Generación de las predicciones:** Se obtienen las 35 predicciones Intra utilizando los vectores de referencia previamente generados.
- **Codificación y decodificación:** Este es el último paso para cada CU y lo lleva a cabo la función `doHEVCMode`, que se encuentra desglosado y explicado más adelante.

C. doHEVCMode

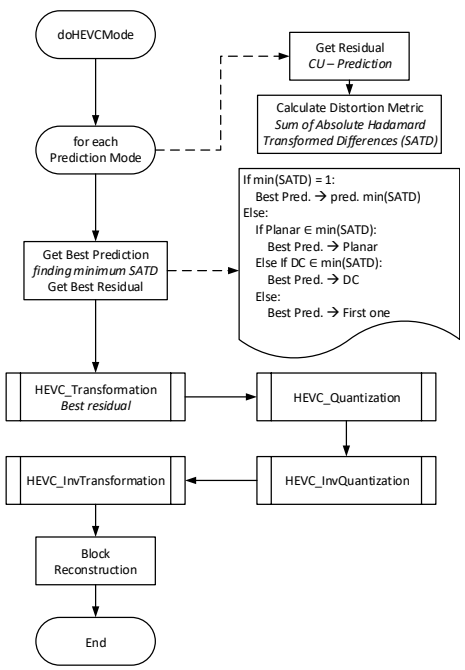


Fig. 5: Diagrama de flujo del archivo `doHEVCMode.m`

En esta función lanzada para cada bloque CU, y cuyo diagrama de flujo se muestra en la Figura 5, se realiza el proceso de cálculo y elección del mejor modo de predicción así como la transformación, cuantización (directa e inversa) del bloque correspondiente.

El primer paso consiste en obtener la mejor predicción posible para el bloque. Como ya hemos visto, en la función `HEVC_Main` se han calculado las 35 predicciones a partir de los vectores de referencia. En esta función se obtiene el residuo de cada modo siguiendo la expresión $CU_{residual} = CU - CU_{predicted}$ y después se calcula su coste o medida de error en base a la suma de las diferencias absolutas transformadas

(en inglés, Sum of Absolute Transformed Differences o SATD).

Este algoritmo, a diferencia de la suma de las diferencias absolutas (SAD), trabaja en el dominio frecuencial aplicando la transformada de Hadamard, computacionalmente más sencilla que la DCT. Debido a que la transformada de Hadamard elimina la redundancia espacial, el uso del SATD es un indicador claro para la distorsión D en el problema de optimización del Rate-Distortion (RDO). El cálculo del SATD para el residuo de un bloque CU de tamaño 4×4 se realiza según la Ecuación 2:

$$\mathrm{SATD} = \frac{1}{2} \sum_{i,j} \left| \mathrm{H} \cdot \mathrm{CU}_{residual} \cdot \mathrm{H}^T \right| \qquad (2)$$

donde H corresponde a la matriz de transformación de Hadamard.

El cálculo del SATD en el emulador lo realiza la función `CalcHad`, donde tiene especifica la transformada para bloques de 4×4 y 8×8 solamente. Para bloques de tamaño superior el cálculo se realiza particionando el bloque en bloques de 8×8, calculando para cada uno el SATD y sumando el resultado de todos.

Una vez se han calculado los SATD para cada modo se selecciona el mejor modo de predicción, que será el que tenga el menor valor de distorsión. Si hay más de un modo con el valor mínimo de SATD la elección del mejor modo se determina en función de los siguientes aspectos:

1. Si el modo Planar se encuentra entre los candidatos se selecciona el modo Planar.
2. Si el modo DC se encuentra entre los candidatos pero no el Planar, se selecciona el modo DC.
3. Si solamente hay modos angulares, se selecciona el de menor índice.

Una vez se ha obtenido el mejor modo de predicción, aplicamos la transformada (función `HEVC_Transformation`) al bloque residuo $\mathrm{CU}_{residual}$, obteniendo la matriz de coeficientes transformados T. Después se aplica la cuantización en la función `HEVC_Quantization`, al cual se le pasa por parámetros T, la matriz de cuantización, el valor QP y la profundidad de bits. Los coeficientes transformados y cuantizados, Q, se guardan para más adelante obtener la entropía del bloque.

Para la reconstrucción del bloque se realiza la cuantización inversa a la matriz Q (función `HEVC_InvQuantization`), obteniendo la matriz IQ. Finalmente, se le aplica la transformada inversa (función `HEVC_InvTransformation`) obteniendo el bloque reconstruido IT.

D. GenSequenceSetMetrics

Realiza el cálculo de las métricas de calidad a partir de las imágenes originales y reconstruidas así como la entropía y guarda estos valores en un fichero de texto. En la Figura 6 se muestra su diagrama de flujo.

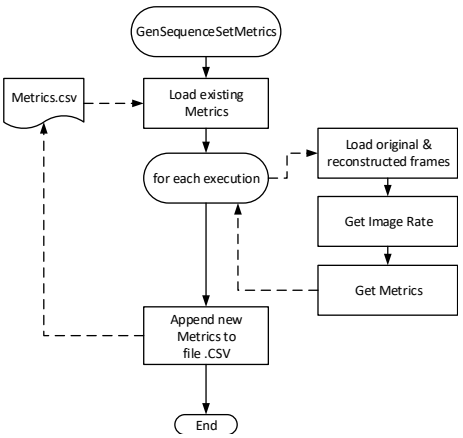


Fig. 6: Diagrama de flujo del archivo `GenSequenceSetMetrics.m`

El primer paso consiste en cargar en memoria el archivo de métricas `Metricas.csv`, si existe, almacenado en el directorio de salida. Después, por cada ejecución, comprueba si ya existe el cálculo de la entropía y de las métricas, en cuyo caso pasa a la siguiente ejecución. Si la ejecución no tuviese el valor previamente calculado, la función realizaría las siguientes acciones:

1. Cargar en memoria el frame original, el reconstruido y la matriz con los coeficientes cuantizados.
2. Obtener la entropía del bloque.
3. Calcular y guardar métricas.

La entropía (el rate) y la calidad nos permitirán obtener las curvas rate-distortion que se obtienen como resultado de la ejecución. El estándar HEVC incluye la codificación entrópica binaria CABAC, pero en este trabajo se ha utilizado el entrópico de Shannon de orden cero por simplicidad para estimar el rate. Para obtener el número de bits totales necesarios para codificar el frame se multiplica la entropía por el número de píxeles del frame. Finalmente, para obtener la tasa de bits multiplicamos los bits por el *framerate* de la secuencia. Se calculan los valores de calidad para las distintas métricas mencionadas anteriormente.

E. genGraph

Este módulo, cuyo diagrama de flujo se muestra en la Figura 7, imprime por pantalla las gráficas para las ejecuciones definidas en los parámetros de la función `EmuladorHEVC`. También genera un archivo CSV con los puntos de las curvas así como un reporte con las métricas Bjøntegaard.

La función carga en memoria el archivo `Metrics.csv` que contiene los parámetros de rate y calidad previamente obtenidos para cada ejecución. Después, dependiendo del valor del parámetro `videomode`, realiza el promedio de todos los *rates* y métricas de calidad para todos los frames de cada secuencia. Para el caso en que `videomode` no esté activo, esta función devolverá una gráfica

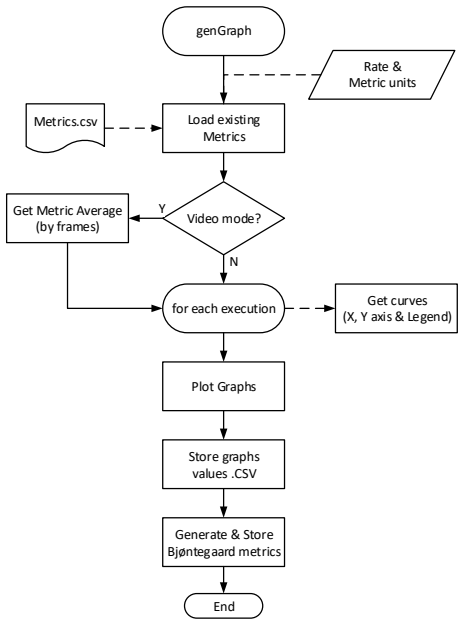


Fig. 7: Diagrama de flujo del archivo `genGraph.m`

para cada uno de los frames que tenga la secuencia.

Las gráficas vienen estructuradas de la siguiente manera: en el eje horizontal se representa el *rate* definido en el parámetro `RateMode`, mientras que en el eje vertical se representa la métrica de calidad, definida en el parámetro `Metric`. Cada tamaño de bloque CU de cada secuencia tiene su propia gráfica independiente, y dentro de cada gráfica se representa mediante curvas las diferentes combinaciones de parámetros restantes (con o sin CSF), donde cada punto de la gráfica corresponde a cada valor del factor de calidad `QPs`.

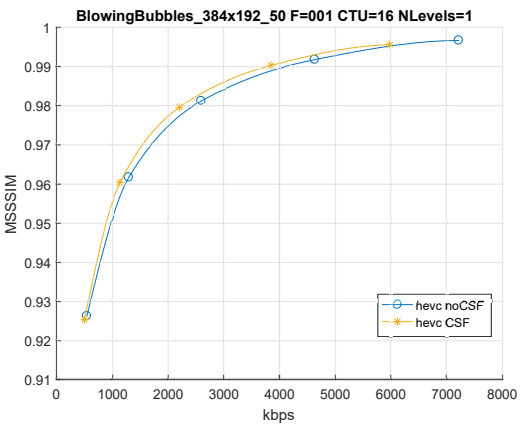


Fig. 8: Gráfica de ejemplo.

La Figura 8 es un ejemplo de ejecución para el frame 1 de la secuencia BlowingBubbles. En el título se muestra el nombre de la secuencia, el número de frame, el valor de CTU y el Nivel de particionado. Esta ejecución se ha lanzado con la opción `videomode` a *false*, en caso de haber sido *true* no aparecería el valor del frame. La leyenda identifica las curvas a partir de los siguientes parámetros: `hevc ␣WeightMode`.

Si se tiene establecido a *true* el valor del campo `print_to_pdf` la ejecución exportará cada una de las gráficas a archivos PDF, asignándoles a cada una un nombre que identifique los parámetros únicos de la misma. Siguiendo el ejemplo de la Figura 8, el nombre del archivo PDF exportado para esa gráfica sería *BlowingBubbles_384x192_50 F=001 CTU=16 NLevels=1 MSSSIM-kbps.pdf*

Con respecto a las curvas, los puntos indican el lugar exacto que relaciona la métrica de calidad y el *rate*, mientras que para la unión de los puntos no se ha trazado una línea recta sino que se ha suavizado mediante interpolación PCHIP (*Piecewise Cubic Hermite Interpolating Polynomial*), algoritmo que incluye Matlab.

Esta función también genera un archivo .CSV con los datos tabulados de una forma específica con el objetivo de ser abierto con una hoja de cálculos (por ejemplo Microsoft Excel) y realizar fácilmente las mismas gráficas con facilidad. En la figura 9 se muestra este archivo para el ejemplo realizado en este apartado.

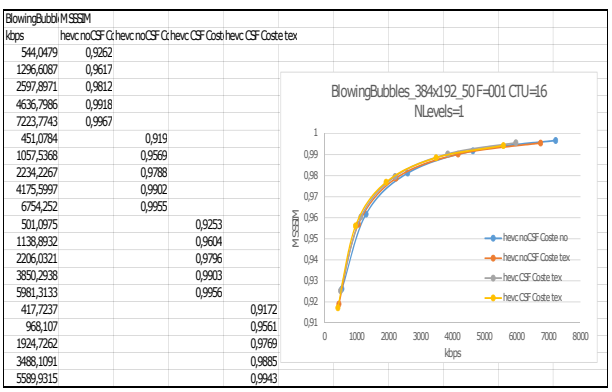


Fig. 9: Archivo CSV generado con los puntos de las gráficas y su representación en Microsoft Excel

Por último, esta función genera otro archivo CSV con el cálculo de las métricas Bjøntegaard, que nos permiten comparar las curvas generadas de forma numérica.

IV. Ejemplos de ejecución

En esta sección mostramos en las figuras 10 a 12 algunos resultados de ejecuciones para distintas secuencias en gráficas Rate-Distortion y valores BD-Rate.

Adicionalmente a las características incluidas en la propia herramienta se ha utilizado la misma para incorporar un modelo de texture masking aplicado al HEVC y presentado en [15], de modo que en las gráficas (ver Figura 13) se puede analizar las mejoras introducidas en el rendimiento del codificador. Así mismo este rendimiento nos lo ofrece la herramienta en formato BD-Rate como vemos en la Tabla II.

V. Conclusiones y trabajo futuro

En este artículo hemos presentado una herramienta emuladora de HEVC en Matlab que facilita la investigación de distintos esquemas de cuantización, transformación, etc. debido a que la Matlab es un

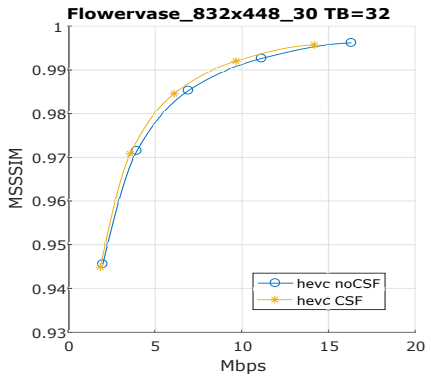


Fig. 10: Gráfica MSSSIM - Mbps para la secuencia *Flowervase* y tamaño de bloque 32×32.

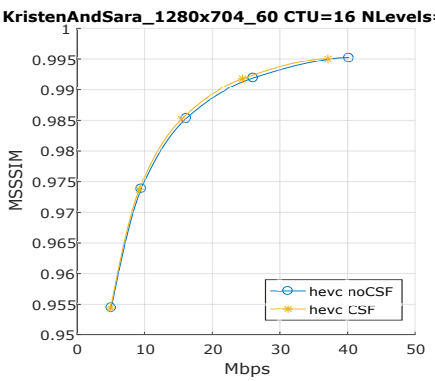


Fig. 11: Gráfica MSSSIM - Mbps para la secuencia *KristenAndSara* y tamaño de bloque 16×16.

lenguaje orientado a matrices y todas las operaciones son más sencillas de analizar e implementar que utilizando el software de referencia. Para ello, esta herramienta ha sido validada contra el software de referencia HM 14.0, garantizando que las etapas de transformación, cuantización y predicción Intra generan los mismos resultados que dicho software de referencia. Así el investigador podrá intercalar y modificar estos esquemas, obteniendo resultados que podrán ser integrados en el software de referencia una vez validados.

La versión actual de la herramienta realiza un particionado de bloques de tamaño fijo, obtiene gráficas

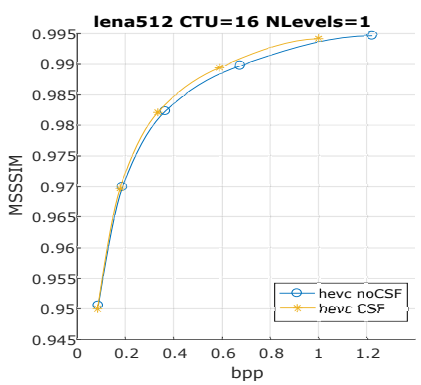
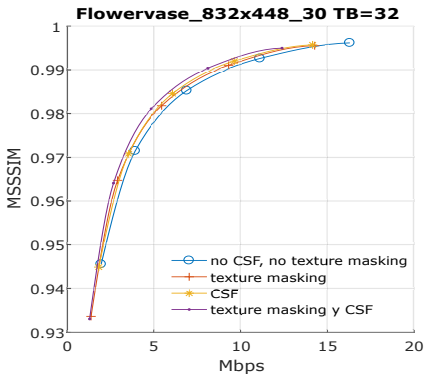
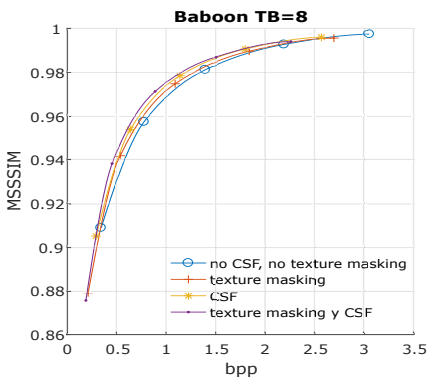


Fig. 12: Gráfica MSSSIM - bpp para la imagen *Lena* y tamaño de bloque 8×8.



(a) Curva MSSSIM - Mbps para tamaño de bloque 32×32



(b) Curva MSSSIM - bpp para tamaño de bloque 16×16

Fig. 13: Ejemplo de curvas Rate-Distortion para diferentes imágenes y secuencias de vídeo.

TABLA II: Resultados BD-Rate (%) para métrica perceptual MS-SSIM y tamaño de bloque 32×32.

Secuencia test	Texture masking	Contrast masking	Texture y contrast masking
Baboon (imagen)	-2.06	-8.90	-10.39
Lena (imagen)	-5.25	-3.81	-9.49
BlowingBubbles	-2.03	-5.01	-6.57
Flowervase	-7.56	-7.56	-13.92
KristenAndSara	-9.09	-2.41	-10.91
Cactus	-6.90	-2.55	-8.84

Rate-Distortion como resultado de su ejecución y calcula valores BD-Rate para las distintas curvas de estas gráficas. También permite habilitar o deshabilitar el uso de matrices de cuantización perceptual.

Como trabajos futuros, planteamos la ampliación de la herramienta con la integración del particionado Quad-Tree del estándar HEVC, la integración del algoritmo CABAC e implementación del RDO en base a él, implementación de los filtros de de-blocking y SAO, la inclusion de nuevas métricas y la implementación de una interfaz de usuario con GUIDE de Matlab.

Esta herramienta estará disponible en breve en su versión actual (y futuras) en la web del grupo de investigación GATCOM [16].

Agradecimientos

Este trabajo ha sido financiado por el Ministerio de Ciencia, Innovación y Universidades con referencia RTI2018-098156-B-C54 cofinanciado con fondos FEDER (MINECO/FEDER/UE).

Referencias

[1] D. Springer, W. Schnurrer, A. Weinlich, A. Heindel, J. Seiler, and A. Kaup, "Open Source HEVC Analyzer for Rapid Prototyping (HARP)," in *IEEE Int. Conf. on Image Processing (ICIP)*, Paris, France, October 2014.

[2] Haoming Chen, Tao Zhang, Ming-Ting Sun, Ankur Saxena, and Madhukar Budagavi, "Improving intra prediction in high-efficiency video coding," *IEEE Transactions on Image Processing*, vol. 25, no. 8, pp. 3671–3682, 2016.

[3] Blaeser M., Rohlfing C., Gao Y., and Wien M., "Adaptive coding of non-negative factorization parameters with application to informed source separation," in *43rd International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Calgary, Alberta, Canada, 2018, Proceedings of the 43rd International Conference on Acoustics, Speech and Signal Processing (ICASSP), IEEE.

[4] Vishal Deep, "Realization of State-of-the-art Intra-Prediction High Efficiency Video Coding (HEVC)," M.S. thesis, Lyles College of Engineering California State University, 12 2016.

[5] Masoud Nouripayam and Nima Sheikhipoor, "HEVC (H.265) Intra-Frame prediction implementation using MATLAB," 2014.

[6] Vivienne Sze, Madhukar Budagavi, and Gary J. Sullivan, *High Efficiency Video Coding (HEVC): Algorithms and Architectures*, Springer Publishing Company, Incorporated, 2014.

[7] Fraunhofer Institute for Telecommunications, "HM Reference Software Version 14.0," `https://hevc.hhi.fraunhofer.de/svn/svn_HEVCSoftware/tags/HM-14.0/`, 2014.

[8] Zhou Wang, A.C. Bovik, H.R. Sheikh, and E.P. Simoncelli, "Image quality assessment: from error visibility to structural similarity," *Image Processing, IEEE Transactions on*, vol. 13, no. 4, pp. 600–612, 2004.

[9] Z. Wang, E.P. Simoncelli, and A.C. Bovik, "Multiscale structural similarity for image quality assessment," in *Signals, Systems and Computers, 2004. Conference Record of the Thirty-Seventh Asilomar Conference on*, 2003, vol. 2, pp. 1398–1402 Vol.2.

[10] H.R. Sheikh and A.C. Bovik, "Image information and visual quality," *Image Processing, IEEE Transactions on*, vol. 15, no. 2, pp. 430–444, 2006.

[11] Nikolay Ponomarenko, Flavia Silvestri, Karen Egiazarian, Marco Carli, Jaakko Astola, and Vladimir Lukin, "On between-coefficient contrast masking of dct basis functions," in *Proceedings of the third international workshop on video processing and quality metrics*, 2007, vol. 4.

[12] Nikolay Ponomarenko, Federica Battisti, Karen Egiazarian, Jaakko Astola, and Vladimir Lukin, "Metrics performance comparison for color image database," in *Fourth international workshop on video processing and quality metrics for consumer electronics*, 2009, vol. 27.

[13] Miguel Onofre Martínez-Rach, *Perceptual image coding for wavelet based encoders*, Ph.D. thesis, Universidad Miguel Hernández, Dec. 2014.

[14] G Bjontegaard, "Calculation of average psnr differences between rd-curves," *Proceedings of the ITU-T Video Coding Experts Group (VCEG) Thirteenth Meeting*, 01 2001.

[15] Javier Ruiz, Otoniel López Granado, Manuel Malumbres, and Miguel Martínez-Rach, "Análisis combinado de texture y contrast masking en hevc," in *Jornadas SARTECO*, 2019.

[16] Universidad Miguel Hernández de Elche, "Grupo de Arquitectura y Tecnología de Computadores," `http://atc.umh.es/gatcom/`.

Redes y comunicaciones

Metodología para la Instrumentación de Simulaciones de Diseños Hardware en SystemVerilog

Juan-Jose Crespo, German Maglione-Mathey, José L. Sánchez, Francisco J. Alfaro-Cortés, Jesus Escudero-Sahuquillo, Pedro Javier García, Francisco J. Quiles [1]

Resumen— **El modelado de hardware utilizando lenguajes de descripción de hardware o HDLs (*Hardware Description Languages*), como *SystemVerilog* se ve limitado por las facilidades que ofrecen esos lenguajes para modelar abstracción de sistemas complejos. Frecuentemente, estos diseños requieren de componentes complejos no necesariamente relacionados con la funcionalidad del producto final, como por ejemplo componentes destinados a la instrumentación o la depuración de errores. Delegar las tareas de instrumentación de diseño hardware a lenguajes de programación de alto nivel, permite a los diseñadores concentrar toda su atención en el propio diseño hardware. Se permite la integración sencilla de modelos a diferentes niveles de abstracción, lo que facilita que modelos existentes desarrollados utilizando lenguajes de programación de alto nivel puedan utilizarse en conjunto con componentes hardware con un menor nivel de abstracción. En este trabajo proponemos una nueva metodología para facilitar la interacción entre componentes de un diseño hardware y componentes externos desarrollados utilizando lenguajes de programación de alto nivel.**

Palabras clave— **systemverilog, hdl, simulación, metodología, instrumentación**

I. Introducción

El diseño y modelado asistido por ordenador de sistemas hardware se ha convertido en una herramienta indispensable al incrementarse la complejidad de estos sistemas. Esto es posible gracias a la reutilización de componentes, incorporando diseños existentes para crear diseños complejos permitiendo mejoras incrementales en cada iteración.

El modelado de hardware también ha experimentado una gran evolución en las últimas décadas. Los lenguajes de descripción de hardware o HDL (del inglés *Hardware Description Languages*), se desarrollaron, y continúan haciéndolo, para simplificar los procedimientos de desarrollo, especificación y verificación de sistemas hardware. Uno de los HDLs más utilizados es SystemVerilog.

Además, se han popularizado plataformas hardware diseñadas específicamente para el modelado y diseño de sistemas hardware. Las mejoras en las tecnologías de fabricación de circuitos integrados ha permitido el desarrollo de dispositivos que permiten implementar un amplio abanico de sistemas digitales. Estos dispositivos se conocen como dispositivos de matriz de puertas programables o FPGAs (del inglés *Field-Programmable Gate Array*). Y desde su introducción en 1985, las FPGAs se han convertido en la plataforma preferida de la mayoría de diseñadores de circuitos digitales.

La tendencia actual en sistemas de altas prestaciones o HPC (*High Performance Computing*), entre otras, es la utilización de sistemas FPGAs, en conjunto con sistemas digitales diseñados utilizando HDLs, para la creación de infraestructuras de cómputo más eficientes [1] [2] [3].

Dentro de este contexto, los diseños hardware interactúan con diferentes sistemas externos, especialmente en la primeras etapas del desarrollo del proyecto. Algunos de estos sistemas externos se modelan utilizando plataformas de simulación. La interacción entre los prototipos y los sistemas externos debe ser concebida de forma modular, que permita que cada sistema coopere de forma desacoplada.

Por otra parte, los arquitectos hardware se enfrentan a proyectos complejos y extensos que consisten en una importante cantidad de módulos que a su vez se componen de módulos más pequeños desarrollados utilizando HDLs. La instrumentación de este tipo de proyecto es en general una tarea compleja, especialmente si uno de los objetivos es el análisis del comportamiento de ciertos aspectos del sistema mediante la instrumentación. Las bibliotecas de verificación de componentes [4] [5] carecen de las características necesarias para instrumentación ya que se orientan a garantizar que la implementación de un diseño sigue estrictamente la especificación, y no el análisis mediante instrumentación.

A pesar de la disponibilidad de herramientas capaces de realizar análisis mediante instrumentación, en general requieren que los arquitectos hardware tengan que reescribir sus diseños utilizando HDLs de alto nivel (e.j. SystemC [6], Hardware-C [7]) que puede producir código que no puede ser sintetizado de forma directa. Además, estos lenguajes están diseñados para ser consistentes con los HDLs existentes, lo que, en algunos casos, dificulta expresar algoritmos complejos. Por ejemplo, SystemC fue diseñado con la premisa de sustituir los HDLs en lugar de complementarlos, lo cual requiere que los componentes sean implementados en este lenguaje desde el principio. Esto nos lleva a plantearnos el uso de estos lenguajes para el modelado de componentes que no tienen como fin ser sintetizados, como por ejemplo componentes que simulan el tráfico de una aplicación o la ejecución de aplicaciones reales (e.j. tráfico

[1]Dpto. de Sistemas Informáticos, Universidad de Castilla-La Mancha, Albacete, España, {juanjose.gcrespo, jose.sgarcia, fco.alfaro, jesus.escudero, pedrojavier.garcia, francisco.quiles}@uclm.es
german.maglione@dsi.uclm.es

del protocolo de coherencia) utilizando herramientas externas como gem5[8].

En este trabajo presentamos un metodología (en desarrollo) y una herramienta diseñada para controlar la interacción entre diferentes componentes escritos tanto con diferentes niveles de abstracción como posiblemente deferentes lenguajes de programación. El objetivo principal de nuestra propuesta es la reusabilidad de los diseños de componentes existentes, ofreciendo un mecanismo para interconectar múltiples componentes, de forma desacoplada, mediante el intercambio de mensajes.

En particular, nos centramos en diseños hardware escritos utilizando SystemVerilog HDL (SV) ya que es uno de los lenguajes de descripción de hardware más utilizados para el modelado y verificación de Circuitos Integrados (IC). A tal efecto, utilizamos la interfaz DPI (*Direct Programming Interface*), provista por SV que permite la interacción entre SV y un lenguaje de programación externo. En el momento de escribir este artículo, el último estándar definido para SV, el IEEE 1800-2017 [9] sólo permite la interacción de SV, utilizando la interfaz DPI, con el lenguaje de programación C (DPI-C). Por tanto, en este artículo se utiliza el lenguaje C como una capa intermedia para *conectar* SV con otros lenguajes.

La interacción entre la capa intermedia (DPI-C) con otros lenguajes de programación (actualmente no soportados por la interfaz DPI) se consigue gracias a la combinación de dos bibliotecas: Protocol Buffers [10] y ZeroMQ [11]. La primera de ellas se utiliza tanto para la serialización de los datos como para la declaración de estructuras comunes para el intercambio de datos entre los diferentes lenguajes. La biblioteca ZeroMQ proporciona varias formas de transmisión de datos: *in-process* (dentro de un mismo proceso), *inter-process* (entre diferentes procesos), *TCP* y *multicast*. Esta biblioteca se encarga de la comunicación entre el sistema que se encarga de la simulación de los modelos escritos en SV y los programas que se comunican con él, en general, escritos en diferentes lenguajes de programación.

Este artículo está organizado de la siguiente manera: En la Sección II se describen los conceptos básicos de las bibliotecas Protocol Buffers y ZeroMQ. La Sección III introduce la metodología propuesta y los mecanismos disponibles para permitir la interacción entre modelos. Un caso de uso de esta metodología aplicada a un diseño de hardware existente se describe en la sección IV. Y finalmente, las conclusiones de este trabajo pueden verse en la Sección V.

II. Protocol Buffers y ZeroMQ

A. Protocol Buffers

Los autores de la biblioteca Protocol Buffers (PB) la describen como un *mecanismo automatizado flexible y eficiente para la serialización de estructura de datos* [10]. La biblioteca PB se utiliza para definir la información que será serializada como mensajes PB (*protocol buffer messages*). Siendo estos mensajes parte de la especificación del protocolo de transmisión. En el caso de la metodología propuesta, los mensajes PB definen la estructura de los datos intercambiados entre diferentes módulos que requieren algún tipo de interacción, que además pueden estar modelados a diferentes niveles de abstracción.

Una de las características de la biblioteca PB es que los mensajes se codifican en formato binario. En particular, los números se representan utilizando una codificación en base 128, conocida como *varint*. El nombre *varint* puede entenderse como enteros de tamaño variable (contracción del inglés *variable sized integer*). Que la biblioteca PB utilice una representación de enteros de tamaño variable permite almacenar los números con el mínimo número de *bytes* necesarios para ser decodificados. Los números enteros codificados como *varint*s utilizan uno o más *bytes* en donde se utiliza el *bit* más significativo (msb) para indicar que los siguientes *bytes* pertenecen o no al mismo número, estableciendo ese *bit* a 1 o 0, respectivamente. Además, los enteros se representan utilizando el *complemento a dos*, dividiéndolos en grupos de 7 *bits*.

Según la descripción dada anteriormente el número de *bytes* necesarios para codificar un número entero n es:

$$\left\lceil \frac{\log_2(n)+1}{7} \right\rceil$$

Adicionalmente, PB permite la existencia de campos opcionales en los mensajes y la definición de mensajes anidados, siendo ambos codificados como parte de la especificación. Todas estas características permiten un diseño modular del protocolo de transmisión, capaz de representar la interacción entre diferentes módulos de forma eficiente.

El Listado 1 muestra un ejemplo de la definición de un mensaje con 3 campos, uno de ellos opcional.

Listado 1: Ejemplo de definición de un mensaje en PB.

```
message pb_message{
    required uint32 field0  = 1;
    required bool field1    = 2;
    optional string field2  = 3;
}
```

Existen otras bibliotecas alternativas aparte de PB que utilizan mensajes codificados en binario, como por ejemplo *Cap'n Proto* [12] y *MessagePack* [13], entre otras. Estas bibliotecas pueden remplazar a PB en el contexto de esta metodología. Sin embargo, en nuestro caso hemos decidido utilizar PB por la familiaridad de los autores con su API.

B. ZeroMQ

Permitir la comunicación entre diferentes componentes ejecutados en procesos separados, que pueden estar, a su vez, en ordenadores interconectados por alguna clase de red, no es una tarea trivial y debe ser planificada con detalle.

En este contexto, es necesaria una herramienta que se encargue de forma automática de los mensajes transmitidos entre los diferentes componentes,

y además de la representación de esos mensajes *en el cable*, simplificando las operaciones de envío y recepción. También es necesario tener en cuenta que algunos componentes pueden ser más lentos que otros tanto en la recepción como en el envío de mensajes.

Existen varias herramientas desarrolladas para eliminar, o al menos simplificar, la complejidad de permitir la comunicación entre diferentes componentes. Así, por ejemplo, *Apache Thrift* [14] permite el desarrollo escalable de servicios posiblemente escritos en diferentes lenguajes de programación utilizando llamadas a procedimientos remotos (RPC). Sin embargo, aún se encuentra en desarrollo y carece de una versión estable. Otra alternativa es el *Advanced Message Queueing Protocol* (AMQP) [15], un protocolo diseñado para el intercambio confiable de mensajes entre dos partes, mediante la utilización de un intermediario o *broker* (*message broker*). Existen varias implementaciones del protocolo AMQP, una de ellas es *Apache Qpid* [16]. Como en el caso de *Thrift*, *Qpid* no dispone de una versión estable, y además las nuevas versiones del protocolo AMQP no mantienen la compatibilidad hacia atrás.

Una vez evaluadas varias alternativas, hemos considerado la biblioteca *ZeroMQ* (ZMQ)[11] que provee un *marco de concurrencia* utilizando *sockets* para permitir la transmisión de mensajes de forma atómica mediante diferentes modos de transporte (in-process, inter-process, TCP, y multicast). Dada su interfaz (API) intuitiva, la variedad de patrones de comunicación listos para ser usados y versiones estables, elegimos ZMQ para ser utilizada como parte de la metodología propuesta.

Una característica importante de ZMQ es la transmisión de mensajes entre aplicaciones sin necesidad de realizar varias copias, un técnica conocida como *zero-copy* (sin copia)[1]. Esta característica puede mejorar el rendimiento de algunas aplicaciones, en particular aquellas que realizan comunicaciones entre procesos (*inter-process*).

La biblioteca ZMQ también ofrece una amplia variedad de patrones de comunicación, como por ejemplo *Request-Reply*, *Publish-Subscribe* y *Pipeline*. El tipo de patrón utilizado depende principalmente de la interacción entre los módulos, eligiendo aquel que mejor se ajuste a las necesidades de comunicación. En este caso, utilizamos el patrón solicitud/respuesta *Request-Reply*.

No debe confundirse el concepto de *socket* en ZMQ con la definición tradicional de *socket* de Internet. ZMQ considera a los *sockets* como puntos de acceso a *motores de comunicación* capaces de manejar múltiples conexiones de forma automática. Por lo tanto, los *sockets* en ZMQ siempre se asocian a un patrón de comunicación habilitado por un *motor de comunicación* que gestiona dicho *socket*. Cabe destacar que esto requiere que se defina el patrón de comunicación en el momento de crear el *socket*. Las comunicaciones entre *sockets* con patrones de comunicación compatibles se realizan de forma transparente por el *motor de comunicación* que implementa ese patrón.

Esta arquitectura permite a ZMQ encargarse de forma automática de situaciones en donde algunos agentes (procesos o aplicaciones) no están disponibles temporalmente. Pudiendo entregar mensajes a estos agentes tan pronto como estén disponibles. Este comportamiento depende del tipo de *socket* utilizado y de las características soportadas.

La Figura 1 muestra las tareas que realiza cada biblioteca en la transmisión de un mensaje.

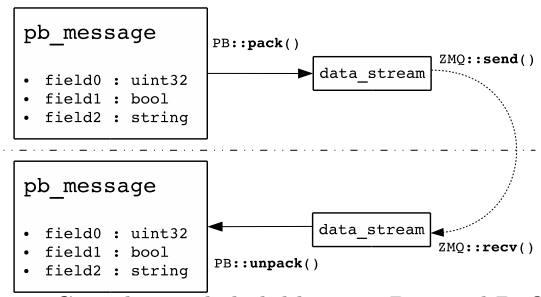


Fig. 1: Caso de uso de la bibliotecas Protocol Buffers y ZeroMQ.

III. Metodología

En esta sección presentamos una metodología diseñada para ayudar a los arquitectos y diseñadores de hardware a integrar, de forma modular, diferentes modelos eliminado la necesidad de gestión de las comunicación generada por la interacción entre módulos.

De forma simplificada, esta metodología necesita de componentes que realicen las siguientes funciones:

1. Interfaz y encapsulación de datos.
2. Gestión de las conexiones.
3. Transmisión y entrega de mensajes.

Un esquema de esos componentes se muestra en la Figura 2, donde se aprecian dos módulos independientes y las interacciones entre ellos. Cabe señalar que esos módulos no necesariamente deben estar en el mismo ordenador, aunque en ese caso el rendimiento puede verse afectado como consecuencia de la cantidad de datos intercambiados y/o la naturaleza de la comunicación entre ambos.

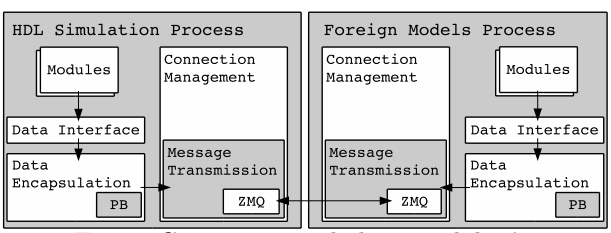


Fig. 2: Componentes de la metodología.

La interfaz de datos sólo es necesaria para aquellos lenguajes de programación que no tengan soporte en la biblioteca *Protocol Buffers* (PB). Un ejemplo de uso de esta metodología está disponible en [17].

La metodología propuesta en este trabajo pretende ser un marco general para permitir la interacción

[1]El *Zero* en el nombre de ZMQ hace referencia a que no es necesario un *broker* o *zero-broker*, y no por no realizar copias

entre diferentes componentes modelados utilizando *SystemVerilog HDL* (SV) con entidades externas que pueden realizar diferentes tareas, que no necesariamente sean parte del diseño, pero sin embargo relacionadas, como por ejemplo la instrumentación, prototipado o depuración.

Por tanto, esta metodología puede ser aplicada a cualquier proyecto de diseño hardware que utilice SV y que requiera interactuar con componentes externos, desarrollados o no en SV, de una forma desacoplada y posiblemente ejecutándose en diferentes procesos/ordenadores.

A. Interfaz y encapsulación de datos

El objetivo de la capa de interfaz de datos es proveer de una *interfaz de programación de aplicaciones* (API) (*Application Programming Interface*) consistente entre diferentes lenguajes, para permitir el envío y la recepción de datos según la especificación, generando, a su vez, la definición de los mensajes necesarios para la comunicación.

La biblioteca *Protocol Buffers* (PB) dispone de las herramientas necesarias para definir la especificación y el formato de los mensajes para diferentes lenguajes de programación. La definición de los mensajes se realiza utilizando los ficheros *.proto*. Un vez especificados, PB se encarga de la generación de las estructuras de datos y ficheros específicos para el lenguaje de programación seleccionado.

No obstante, PB no contempla ningún lenguaje para diseño hardware (HDL), por lo tanto, es responsabilidad del programador facilitar una interfaz para que los lenguajes no soportados por PB puedan interactuar con esta biblioteca. En este trabajo hacemos uso de PB desde *SystemVerilog* (SV) utilizando la interfaz DPI-C.

Las estructuras de datos necesarias para implementar la interfaz de datos en SV utilizando DPI-C se muestran en la Figura 3. Para permitir que los módulos escritos en SV sean capaces de enviar y recibir datos, es necesario que se definan las estructuras de datos adecuadas al tipo de interacción que los módulos requieren. Con este objetivo, se debe definir un conjunto de funciones (conformes a la definición de la interfaz DPI-C), tanto para SV como para el lenguaje C, que los módulos puedan *llamar* de forma directa. Tanto las funciones, definidas arriba, como las estructuras de datos pertenecen a los bloques de interfaz y encapsulación de datos, como se muestra en la Figura 3.

Los datos son accedidos/provistos por los módulos al bloque de encapsulación, a través del bloque de interfaz de datos. El siguiente paso consiste en des/encapsular estos datos convirtiéndolos a mensajes PB. Este paso es trivial ya que tanto los mensajes PB como las estructuras de datos definidas en el bloque de interfaz de datos comparten la misma *estructura* (es decir, tienen el mismo número y tipo de campos).

Completado el paso anterior, la encapsulación de los datos en mensajes PB, es posible enviar esos mensajes al bloque de transmisión de mensajes para que

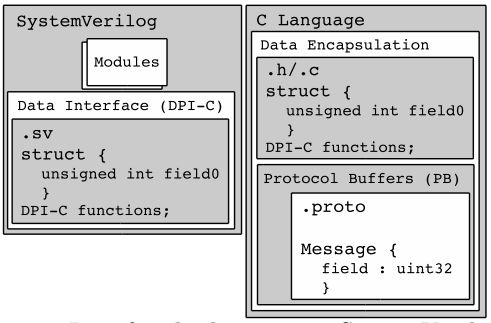


Fig. 3: Interfaz de datos para *SystemVerilog*.

la biblioteca ZMQ pueda transmitirlos utilizando el patrón de comunicación elegido.

B. Gestión de las conexiones

Las conexiones entre los diferentes módulos se gestionan desde este bloque. Estas conexiones involucran a múltiples módulos interactuando entre ellos, posiblemente utilizando diferentes patrones de comunicación. En la Sección II-B ya se explicaron los *sockets* ZMQ, donde cada uno de estos *sockets* están asociados a un patrón de comunicación. Por lo tanto, puede se necesario definir diferentes *sockets* para diferentes módulos según al patrón de comunicación requerido para la interacción con módulos remotos. Esto debe ser gestionado de forma transparente, automatizando la gestión de los *sockets* y sus patrones de comunicación asociados para que los requisitos de comunicación de cada módulo sea satisfecha.

La Figura 4 muestra el bloque correspondiente a la gestión de las conexiones donde se incluyen el conjunto de patrones de comunicación.

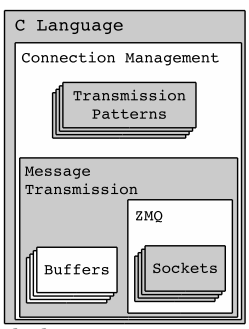


Fig. 4: Gestión de las comunicaciones y transmisión de mensajes.

Los patrones de comunicación provistos por ZMQ[11] son:

- *Request-Reply*: Solicitud/respuesta, conecta un conjunto de *clientes* con un conjunto de *servidores*.
- *Publish-Subscribe*: Publicación/Suscripción, conecta un conjunto de *publicadores* con un conjunto de *suscriptores*
- *Pipeline*: Conecta un conjunto de nodos en un patrón *fan-out/fan-in*.

Dentro de un mismo patrón de conexión, cada *socket* toma diferentes roles, algunos de los cuales se enumeran a continuación.

- *Request-Reply*: {`REQ, REP`}
- *Publish-Subscribe*: {`PUB, SUB`}
- *Pipeline*: {`PUSH, PULL`}

El rol que cumple cada *socket* debe tenerse en consideración cuando se gestionan las conexiones para permitir (o no) transferencias bidireccionales entre módulos satisfaciendo las restricciones del patrón de conexión.

La Figura 5 muestra un ejemplo de utilización del patrón *Request-Reply* entre dos módulos. En este ejemplo ambos módulos necesitan un par de *sockets* ZMQ `REQ-REP` para que tanto el módulo A como el B puedan iniciar una comunicación. Si se utilizara un único par de *sockets* `REQ-REP` solo se permitiría iniciar la comunicación (*request*) al módulo cuyo *socket* tenga el rol `REQ`. Por lo que el módulo cuyo *socket* tenga el rol `REP` estará restringido a responder (*replay*) y no tendrá permitido iniciar ningún tipo comunicación con otro módulo. El módulo con el *socket* `REQ` puede identificarse con un *cliente*, mientras que el módulo con el *socket* `REP` puede identificarse con un *servidor*.

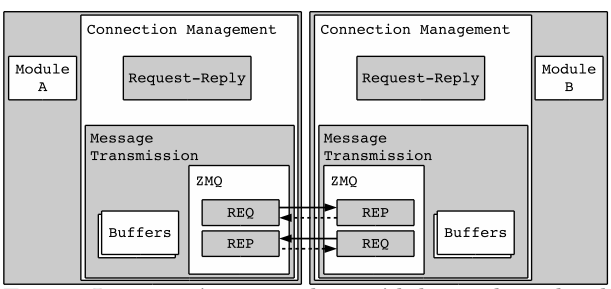


Fig. 5: Interacción entre dos módulos utilizando el patrón *Request-Reply*.

C. Transmisión y entrega de mensajes

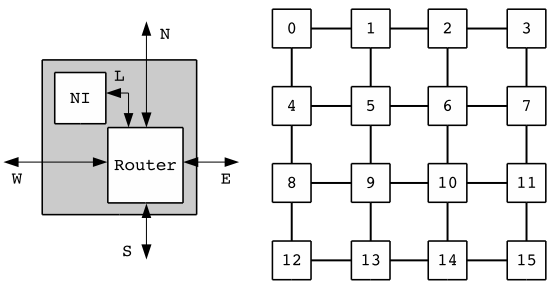
La biblioteca ZMQ se encarga de forma automática de la transmisión de mensajes a través de *sockets*. Sin embargo, para que la transmisión pueda ser realizada ZMQ debe tener acceso a un espacio de memoria (*buffer*) previamente reservado, como muestra la Figura 4. Cada uno de estos *buffers* (nos referimos a *buffers* en memoria y a *Protocol Buffers*) se asigna a cada *socket* y son administrados por el bloque de transmisión y entrega de mensajes.

Algunos patrones de comunicación permiten que las operaciones de envío (*send*) y/o recepción (*receive*) de mensajes se realice de forma *no bloqueante*, como es el caso de los patrones *Publish-Subscribe* y *Pipeline* En caso de que se utilice alguno de estos patrones, se exhibe este comportamiento al programador de forma explícita mediante procedimientos/-funciones de envío/recepción específicas.

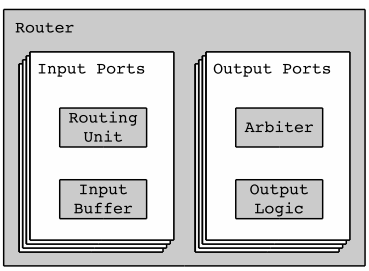
IV. Caso de uso: PEAK Director

En esta sección se introduce un caso de uso real de esta metodología. Se ha aplicado a una arquitectura hardware existente, en particular al subsistema de red de la arquitectura PEAK[18]. Los componentes descritos en esta sección, tanto los modelados utilizando *SystemVerilog HDL* como entidades externas, son específicas de este caso de uso y de los objetivos de instrumentación del proyecto. Estos componentes hacen uso de la metodología propuesta en este trabajo, lo que les permite intercambiar la información necesaria para su funcionamiento.

La arquitectura PEAK define un conjunto de celdas interconectadas formando una malla 2D. Cada una de estas celdas permite que los paquetes de datos puedan ser encaminados por la red, desde/hacia celdas contiguas, además contienen una Interfaz de Red o NI (*Network Interface*) posibilitando la inyección y recepción de paquetes de datos. La Figura 6a muestra un esquema simplificado del subsistema de red en PEAK formando una malla 4x4.



(a) Ejemplo de una topología *mesh* 4x4.



(b) Diagrama de bloque del *Router*.

Fig. 6: Subsistem de red en PEAK.

El objetivo principal de la aplicación de esta metodología es permitir la inyección/entrega de paquetes de datos desde/a una entidad externa, pero utilizando el módulo NI de cada celda. Esta entidad externa puede ser un generador de tráfico sintético, un generador de tráfico basado en trazas, etc., que envía paquetes de datos utilizando la NI y recibe los paquetes entregados por la NI a cada celda. Además, esta entidad está desarrollada en un lenguaje de programación de alto nivel Python, diferente al lenguaje utilizado en PEAK y se ejecuta en un proceso independiente. A esta entidad la hemos llamado *Director* PEAK.

En este caso utilizamos el patrón de comunicación *Request-Reply*. Cada módulo NI, en cada celda, solicita un paquete para ser enviado al generador de tráfico cada vez que está disponible para enviar, es decir, cuando el *buffer* de entrada del puerto local tiene espacio suficiente. El generador de tráfico responderá con un paquete de datos por cada solicitud. La Figura 7 ilustra este procedimiento.

Los módulos que componen los *router*s están desarrollados utilizando *SystemVerilog HDL* a nivel de transferencia de registro o RTL (*Register-transfer*). Este diseño se corresponde con un *router* segmentado

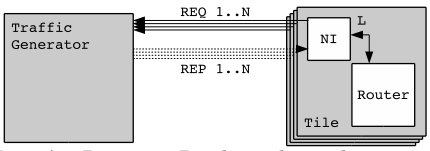


Fig. 7: Patrón *Request-Reply* utilizando un generador de tráfico externo.

de 6 etapas mostrado en la Figura 6b. Por claridad se han omitido algunos componentes, pero se puede consultar [18] para más detalles. También, los *buffer*s de entrada, árbitros y la lógica de salida envían datos estadísticos utilizando un tipo de mensaje especial. Tras recibir estos datos en el *Director* PEAK pueden ser almacenados en una base de datos y/o analizados a posteriori. Los paquetes de datos estadísticos no requieren de respuestas (*replay*) a menos que deba tomarse alguna acción, como por ejemplo, aplicar alguna estrategia para la gestión de la congestión.

En este caso, tanto para el generador de tráfico externo como los *router*s sólo es necesario un único par de *socket*s *REQ-REP*. Esto se debe a que el subsistema de red en PEAK solicita nuevos paquetes tras la inyección/entrega utilizando los módulos NI. Además, la información de los componentes del *router*, como por ejemplo la ocupación de los *buffer*s, estado de los árbitros, etc., sólo se envía cuando es necesario. Por lo tanto, las transferencias de datos siempre se inician desde *SystemVerilog HDL* hacia el *Director* PEAK. Un esquema simplificado de la arquitectura del *Director* PEAK se muestra en la Figura 8.

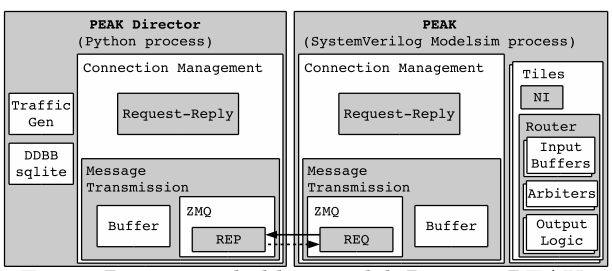


Fig. 8: Diagrama de bloques del *Director* PEAK.

V. Conclusiones

En este trabajo presentamos una metodología diseñada para solventar las limitaciones impuestas por los lenguajes HDLs en el diseño de hardware. Estas limitaciones se manifiestan especialmente cuando se diseñan componentes para generar estímulos complejos a los componentes del diseño.

Además, esta metodología permite a los diseñadores de hardware *conectar* mecanismos para instrumentación y extracción de datos minimizando el impacto en el diseño. Por lo tanto, esta metodología puede ser utilizada en cualquier tipo de diseño hardware que utilice *SystemVerilog HDL* para desacoplar las tareas de instrumentación y depuración de errores del diseño principal. También permite el prototipado de mecanismos complejos, modelando dichos mecanismos con herramientas externas y permitiéndoles interactuar con el diseño hardware.

Por otra parte, aplicamos y presentamos un caso de uso de esta metodología general a un diseño hardware existente, lo que nos permitió desarrollar una herramienta sofisticada que nos ayuda a analizar de forma exhaustiva el subsistema de red de la plataforma PEAK, siendo capaces de generar patrones de tráfico complejos, recoger datos estadísticos y aplicar técnicas de prototipado sin tener que utilizar construcciones *SystemVerilog HDL*. Muchas de las técnicas de gestión de congestión, particionado de la red, reconfiguración dinámica, etc. se desarrollan dentro de *Director* PEAK, y una vez comprobado que son efectivas, el siguiente paso es implementarlas utilizando construcciones HDL tanto a nivel *Behavioral* o *Register-transfer* (RTL).

Agradecimientos

El presente trabajo ha sido financiado conjuntamente por el Ministerio de Ciencia, Innovación y Universidades del Gobierno de España y fondos FEDER de la Comisión Europea, a través de los proyectos TIN2015-66972-C5-2-R y RTI2018-098156-B-C52, la Junta de Comunidades de Castilla-La Mancha a través del proyecto SBPLY/17/180501/000498, Universidad de Castilla-La Mancha y fondos FEDER bajo el marco del proyecto 2019-GRIN-27060. Juan-Jose Crespo es financiado por MECD del Gobierno de España mediante la beca nacional FPU FPU15/03627. German Maglione-Mathey es financiado por la Universidad de Castilla-La Mancha (UCLM) mediante beca pre-doctoral del plan propio de la UCLM PREDUCLM16/29. Jesus Escudero-Sahuquillo es financiado mediante un contrato de acceso al sistema español de ciencia, tecnología e innovación (SECTI) de la Universidad de Castilla-La Mancha y del la Comisión Europea (FSE funds) (UCLM resolucióncon fecha 31/07/2014).

Referencias

[1] Domenico Argenziano, "A research project on fpga-accelerated cryptographic computing," in *P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC), 2015 10th International Conference on*. IEEE, 2015, pp. 574–577.

[2] Alessandro Cilardo and Domenico Argenziano, "Securing the cloud with reconfigurable computing: An fpga accelerator for homomorphic encryption," in *Proceedings of the 2016 Conference on Design, Automation & Test in Europe*. EDA Consortium, 2016, pp. 1622–1627.

[3] José Flich, Giovanni Agosta, Philipp Ampletzer, David Atienza Alonso, Carlo Brandolese, Alessandro Cilardo, William Fornaciari, Ynse Hoornenborg, Mario Kovač, Bruno Maitre, et al., "Enabling hpc for qos-sensitive applications: the mango approach," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2016*. Ieee, 2016, pp. 702–707.

[4] Inc Accellera Organization, "Open Verification Library," 2009.

[5] Inc Accellera Organization, "Universal Verification Methodology," 2012.

[6] Preeti Ranjan Panda, "Systemc: a modeling platform supporting multiple design abstractions," in *Proceedings of the 14th international symposium on Systems synthesis*. ACM, 2001, pp. 75–80.

[7] Rajesh Kumar Gupta, *Co-synthesis of hardware and software for digital embedded systems*, vol. 329, Springer Science & Business Media, 2012.

[8] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R Hower, Tushar Krishna, Somayeh Sardashti, et al., "The gem5 simulator," *ACM SIGARCH Computer Architecture News*, vol. 39, no. 2, pp. 1–7, 2011.

[9] SystemVerilog Language Working Group, "IEEE Standard for SystemVerilog–Unified Hardware Design, Specification, and Verification Language," *IEEE Std. 1800-2017*, 2017.

[10] Google, "Protocol Buffers. Google's Data Interchange Format," 2008, https://developers.google.com/protocol-buffers.

[11] Pieter Hintjens, *ZeroMQ: messaging for many applications*, .ºReilly Media, Inc.", 2013.

[12] K Varda, "Cap'n proto: Introduction," 2013, https://capnproto.org/index.html.

[13] Sadayuki Furuhashi, "Messagepack," 2013, https://msgpack.org/index.html.

[14] Apache Software Foundation, "Apache thrift," 2007, https://thrift.apache.org/index.html.

[15] Steve Vinoski, "Advanced message queuing protocol," *IEEE Internet Computing*, , no. 6, pp. 87–89, 2006.

[16] Apache Software Foundation, "Open source amqp messaging," 2013, https://qpid.apache.org/index.html.

[17] Juan-Jose Crespo, "High level modeling integration for hardware description languages," 2019, https://gitlab.com/jjgcc/hlmhdl.

[18] José Flich, Giovanni Agosta, Philipp Ampletzer, David Atienza Alonso, Carlo Brandolese, Etienne Cappe, Alessandro Cilardo, Leon Dragić, Alexandre Dray, Alen Duspara, et al., "Exploring manycore architectures for next-generation hpc systems through the mango approach," *Microprocessors and Microsystems*, vol. 61, pp. 154–170, 2018.

Node-type-based load-balancing routing for Parallel Generalized Fat-Trees

John Gliksberg[1], Jean-Noël Quintin[2], and Pedro Javier García[3]

Abstract— **High-Performance Computing (HPC) clusters are made up of a variety of node types (usually compute, I/O, service, and GPGPU nodes) and applications don't use nodes of a different type the same way. Resulting communication patterns reflect organization of groups of nodes, and current optimal routing algorithms for all-to-all patterns will not always maximize performance for group-specific communications. Since application communication patterns are rarely available beforehand, we choose to rely on node types as a good guess for node usage. We provide a description of node type heterogeneity and analyse performance degradation caused by unlucky repartition of nodes of the same type. We provide an extension to routing algorithms for Parallel Generalized Fat-Tree topologies (PGFTs) which balances load amongst groups of nodes of the same type. We show how it removes these performance issues by comparing results in a variety of situations against corresponding classical algorithms.**

Keywords— **HPC, routing, fat-tree, heterogeneity**

I. Introduction

ROUTING algorithms for HPC systems aim, for one thing, to avoid congestion during application execution. No perfect agnostic algorithm exists, and designing a good routing algorithm usually requires paying attention to the topology and communication patterns which will take place. The topology of an HPC cluster rarely changes in general (if we don't consider existing equipment disconnection), so algorithms are usually designed for a given topology class (i.e. topology-aware algorithms). On the other hand, communication patterns are hard to observe (it is distributed information and can evolve rapidly), rarely known in advance (sometimes unpredictable, sometimes not predicted, sometimes classified), and in the case of multiple applications running concurrently it is potentially impossible to reroute the cluster on-the-fly for optimal performance of each application without causing deadlocks; indeed, there are few algorithms which rely on real communication patterns. Existing research instead focuses on common worst case scenarios: scatter, gather, n2pairs, all-to-all, hot-spots, etc.

We observe that when nodes are not all of the same type (compute, I/O, service, GPGPU, ...) communication patterns for applications will usually be subsets of worst-case scenarios with only one type of source nodes and one type of destination nodes. In the case of parallel generalized fat-trees (PGFTs), it is intuitive to observe how existing load-balancing algorithms (namely Dmodk and Smodk) can result in avoidable congestion during type-specific communication phases. The aim of this article is to propose new routing algorithms which will provide the same performance for type-specific patterns as existing ones do for type-unspecific patterns.

Section II describes the existing context in detail to improve understanding of the performance issues. Characteristics of fat-tree topologies and their routing algorithms are presented to ease analysis of routing algorithms' quality. A case-study to this effect is introduced alongside a description of node type heterogeneity in Section III. A corresponding communication pattern is chosen in Section IV, alongside a statically-computed congestion metric; three routing algorithms are then analysed in detail to show how they under-utilize available network resources. Section V provides a new technique to use these resources more efficiently without losing properties of the existing algorithms.

II. Background

A. Types of fat-tree topologies

Fat-tree topologies were introduced by Leiserson [1] for their high capacity to represent any network for a given size. All fat-tree topologies are deadlock-free when routed with shortest paths; this property is one of the main advantages of fat-trees. K-ary n-trees were subsequently formalized by Petrini [2] and describe real life implementations of fat-trees with low-radix switches while being rearrangeably-non-blocking for any n2pair pattern. Extended Generalized Fat-Trees (XGFTs), introduced by Ohring [3], describe a more general class of topologies which keep some properties of k-ary n-trees but allow building much smaller and cheaper networks with only partly reduced overall performance. These topologies are generally not capable of providing full cross-bisectional bandwidth (CBB) for a given number of end-nodes and switch radix.

Zahavi introduced Parallel Generalized Fat-Trees which can provide slimmed topologies with full CBB [4]; they are also useful for their fast tolerance to faults on duplicated links. PGFTs are defined by their number of levels h, the upwards arity w, downwards arity m, and link parallelism p, each parameter being defined for every level in the topology. (The corresponding function is $PGFT(h; m_0, \ldots, m_{n-1}; w_0, \ldots, w_{n-1}; p_0, \ldots, p_{n-1})$.) They are built recursively with duplicated subgroups composed of smaller PGFTs (containing only

[1]Laboratoire d'informatique, parallélisme, réseaux, algorithmes distribués (Li-Parad, UVSQ, Versailles, France), Redes y arquitecturas de altas prestaciones (RAAP, UCLM, Albacete, Spain), High-level bull exascale interconnect (BXIHL, Atos, Bruyères-le-Châtel, France), email: john.gliksberg@uvsq.fr.

[2]Atos, France, email: jean-noel.quintin@atos.net.

[3]Redes y arquitecturas de altas prestaciones (RAAP, UCLM, Albacete, Spain), email: pedrojavier.garcia@uclm.es.

interconnected switches of lower levels). Each switch is addressed with a tuple $(l, a_h, \ldots, a_l)$ where l is its level and a is the vector describing the sub-trees at which the node is located.

B. Vocabulary

Fat-trees are composed of levels consisting of switches connected above and below. As a result, elements related to a switch can be classified using an *up* or *down* prefix. For example *up-switches* are switches linked to the switch in question which are in the level above; the same logic applies to its ports and links.

Up and *down*-routes also relate to the direction level-wise. It is worth pointing out that this differs from existing topology-agnostic Up/Down routing algorithms, where *up* means "towards the root node" and conversely for *down*.

Top-switches are those in the highest level; *leaf*-switches are (as in all indirect topologies) those connected to end-nodes. For fat-trees we call all switches in the lowest level *leaves*.

Hereafter, *L1* switches are those at the first level (leaves), followed by *L2*s, etc.

C. Side-note on adaptive routing

Adaptive routing can react to congestion by diverting communication towards alternative routes. In the case of network congestion, spreading congested traffic as much as possible is a right approach, because the congestion was caused by unnecessarily colliding traffic flows in the first place.

Congestion can, however, originate from end-nodes themselves (it can then be referred to as end-node congestion), in that case spreading congested traffic will not solve the situation and will instead further increase the problems arising from that traffic to the rest of the topology. In particular, more traffic flows will share paths with congested traffic, hence increasing the probability of the latter causing head-of-line blocking to the former [5]. Furthermore, traffic that was routed adaptively loses the property of being transmitted in-order, potentially causing supplementary cost to the application or communication layer; for that reason the communication layer can mark packets to forbid them from being routed adaptively.

Since alternative routing cannot differenciate between end-node congestion and network congestion, it does not undermine the need for high-quality deterministic routing. Instead, research focuses on techniques to either reduce the potentially harmful collateral impact of adaptive routing or reduce congestion from happening in the first place with injection-throttling [6] or better deterministic routing. The latter is the aim of this article.

D. Overview of routing algorithms across fat-tree topologies

Pure fat-trees are never used in supercomputers, because they rely on very high-radix switches when there are many nodes. However if we were to route them, that would simply mean following the single shortest path available between any two nodes. Any pair of nodes is associated with a single switch at an equal and minimal distance from both, it is called the nearest common ancestor (NCA).

When routing k-ary n-trees, every pair of nodes has multiple NCAs. Optimal routing then comes down to distributing NCAs via which to route to avoid network-congestion from happening in the first place. For XGFTs and PGFTs, the problem is extended to take into account per-level arities and parallel links.

D.1 Random routing

When multiple NCAs are available in fat-tree topologies, one approach to balancing the load of deterministic routes is to randomly choose upwards routes. There is only one downward route from a switch to a node. However in PGFT topologies there is a choice among parallel links. This choice is done randomly too.

On average, the routes are randomly load-balanced: all-to-all traffic will not cause implicit bias towards any part of the topology. Deviations from the average will, however, cause routes to overlap and induce network congestion.

D.2 Dmodk routing

The regular structure of fat-tree topologies can be used to uniformly distribute routes and achieve load-balancing routing with a deterministic method based on packet destination ID, instead of random routing. Zahavi defines such a routing algorithm for PGFTs in a closed form with upwards routes P^U leading to destination d for all switches in level l computed as follows [4]:

$$P_l^U(d) := \left\lfloor \frac{d}{\prod_{k=1}^{l} w_k} \right\rfloor \bmod (w_{l+1} p_{l+1})$$

This formula assigns an index among the switch's *up-ports*, which must be indexed beforehand to match the topological addressing scheme. All switches in a level that are not in the same subgroup as the destination are assigned the same upwards route. (Those that *are* in the subgroup must be routed downwards.) This formula and corresponding algorithm are called D-mod-k or Dmodk. This method can be simplified for fat-trees simpler than PGFTs and in all cases balances the load while concentrating routes to the same destination, thus concentrating the undesired effects of same-destination end-node congestion within a single-root subtree.

In the case of PGFTs, parallel links are indexed in a round-robin manner so that all *up-switches* are assigned a route before multiple routes are assigned towards a single switch (via those multiple links). This, combined with the above formula, ensures a repartition similar to the one defined sequentially in Zahavi's previous work concerning fat-trees [7].

Gomez [8] routes k-ary n-trees with a method which applies bitmasks to the destination number. This

method is defined in detail for $k = 2$; a similar approach can be extended for higher values of k. This algorithm can be considered as a specialized version of Dmodk routing.

D.3 Smodk routing

If switches can determine the sources of messages, routing algorithms can use that information as well. From this an alternative to Dmodk can be defined: Smodk, which propagates messages similarly to Dmodk but based on source node ID rather than destination ID. This algorithm concentrates together routes from the same source, thus concentrating the undesired effects of same-source end-node congestion as much as possible.

In cases where communications are symmetrical between patterns with several destinations per source and those with several sources per destination, there is no reason for Dmodk or Smodk to be better than the other. Otherwise there isn't necessarily one choice which is always better, but choosing Smodk for multiple-destination heavy patterns (and Dmodk for multiple source heavy patterns) is a reasonable heuristic [9].

We refer to Dmodk and Smodk together as a class of algorithms as Xmodk for the rest of this article.

III. Heterogeneous clusters

Supercomputers are often clusters made of several types of nodes, rather than the common description of a single type of computing nodes. Other types of nodes can include IO nodes for short and long-term data storage; service or management nodes for login, node reservation, deployment, monitoring, fault-tolerance; GPGPU and FPGA nodes for optimized computations

There are various strategies to place secondary nodes (IO or service) in existing clusters, which are usually not described in research material. In the case of fat-trees, strategies can include:

- Placing a constant number of secondary nodes of each type at every leaf
- Adding an irregular subgroup with secondary nodes connected to the top switches like the other regular subgroups (this generally breaks fat-tree properties)
- Connecting the cluster to an external topology via routers. For example if using a Lustre file system, Lustre routers can be nodes of the cluster leading to an array of IO servers of which the fabric management and routing algorithm are not aware.

As a concrete example, BXI[1] switches have 48 ports. Some BXI switch have only copper ports, some others have three optical ports. The optical ports are placed identically on all switches and are dedicated to nodes physically far within the topology (i.e. management nodes and IO proxy nodes).

[1]BXI is the interconnect technology developped by Bull/Atos. It comprises hardware (switches, links) and software (firmware, low-level and high-level development environment on which are built the fabric management and routing algorithms).

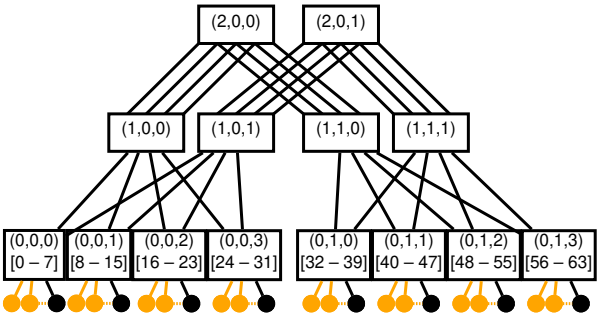


Fig. 1. Case-study topology $PGFT(3; 8, 4, 2; 1, 2, 1; 1, 1, 4)$ — IO nodes (in black) have the largest NID of every leaf $(7, 15, 23, \cdots \equiv 7 \bmod 8)$

IV. Analysis of a type-based communication pattern

We will use a simplified case-study to show how node-type-oblivious load-balancing routing can result in unnecessary network congestion. The topology for this case-study is a pruned 3 level PGFT with low-radix switches and nonfull cross-bisectional bandwidth (CBB) (see figure 1). Nodes are indexed by port rank on their leaf and by leaf address comparison between leaves. The last port of every leaf is reserved for IO nodes; they have NIDs whose modulo by 8 is 7. We use a topology with nonfull CBB because otherwise there would be no possible congestion at any top-port.

This case-study is based on a communication pattern commonly found in distributed applications: data collection from all compute nodes to IO nodes, each compute node sending to the IO node of its symmetrical leaf (e.g.: $(0, 0, 1)$ is symmetrical to $(0, 1, 1)$, so NIDs 8 to 14 send to NID 47). In this case all routes will have to go through a top-switch. This might be contained in a short time frame, following a barrier, or spread out through the application lifetime, it does not matter. For a given complete set of routes R, we call $C2IO(R)$ the subset of routes affected by this pattern.

A. Static congestion metric

This study relies on a static metric to describe the potential sources of contention. The aim of this metric is to give a formal way to describe contention, which abstracts the fine-grain causes of latency to help build a general understanding of how to avoid contention. This contrasts with common techniques based on simulation or experimentation which do not link observations of contention with a corresponding explanation. This simple technique of estimation of contention is new; it is also used in concert with the architecture described by Vigneras [10] with the goal of automating computation of that metric for potential integration into the fabric management's decision making. Analysis in terms of this metric is sufficient to prove and explain drawbacks and benefits of algorithms, but a simulation-based analysis would

complement this work to give tangible results for real-life applications.

Worst-case scenario contention can be measured by the number of possible flows going through a port at the same time. Once the topology is routed, if a given port p is used as output for routes, we can count the number of distinct sources ($src(R,p)$) and destinations ($dst(R,p)$) for these routes:

- Both values are non-nil, since there are routes going through the port.
- If one of these values is equal to one, this port will never be used for unrelated communications; the port is subjected to only one flow of communication. Any potential packet concurrency at the port means that there is a corresponding concurrency at the sending end-node or receiving end-node. That end-node will be the cause of unavoidable congestion of an order of magnitude more important, and no packet from another flow could be affected.

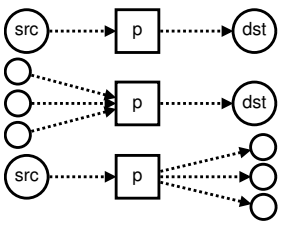


Fig. 2. Example sets of routes for which p is subjected only to one flow

- If both values are greater than one, there are unrelated communications that might interact at this port. This can lead to potentially avoidable network congestion.

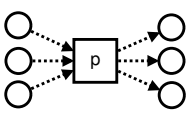


Fig. 3. Example routes for which p can be subjected to multiple flows

For a given set of routes R, we call this metric $\boldsymbol{C}_{port}$:

$$\boldsymbol{C}_p(R) := \min(src(R,p), dst(R,p))$$

This metric does not imply there will always be network congestion at ports with $\boldsymbol{C}_{port} > 1$, but it shows the worst case. Assuming all flows are similar, collisions will happen more frequently when more flows are involved. As a result we claim that in general port a will tend to be more congested than port b if $\boldsymbol{C}_a > \boldsymbol{C}_b$, even if it depends on the exact timing of communications. From this we can deduce a reasonable metric for the whole topology:

$$\boldsymbol{C}_{topo}(R) := \max_{p \in topo} (\boldsymbol{C}_p(R))$$

Routing in a balanced manner means minimizing that metric. Studying a communication pattern means applying this metric only to the routes affected by the pattern rather than all the routes computed.

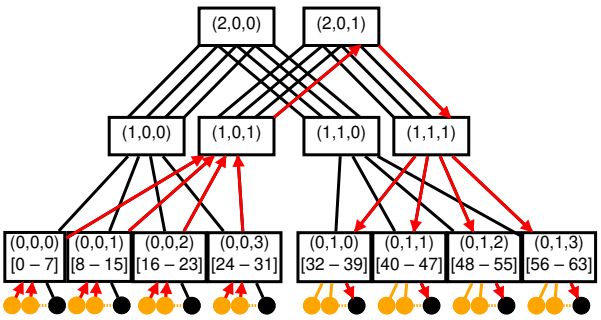


Fig. 4. Set of all routes (in red) going towards IO nodes of the right subgroup, under Dmodk routing. $(2,0,1)$'s port with highest rank is used as output for all routes.

For this metric we consider ports as output for the routes, but the same analysis can be made with ports as input. This does not cause $\boldsymbol{C}_{topo}(R)$ to vary when the pattern has symmetrical communications between sources and destinations.

B. Dmodk performance

With Dmodk routing, destinations will be assigned one switch through which to route in every subgroup. More specifically, we will describe which up-port is routed as output for switches not directly above the destination, and which down-port is routed as output for switches directly above, when there are several parallel ports available.

$w_1 = 2$, $p_1 = 1$: every destination is assigned the L2 switches corresponding to its index modulo two. (E.g.: $47 \bmod 2 = 1$, thus destination 47 is assigned the second L2 switch of each subgroup.) The eight IO destinations are all assigned the same two L2 switches ($(1,0,1)$ and $(1,1,1)$), and more specifically the last up-port of the L2 switch not in their subgroup.

$w_2 = 1$: there is only one L3 switch each L2 switch leads to. It still corresponds to the destination's index modulo 2. IO destinations are assigned the second L3 switch.

$p_2 = 4$: they are more specifically assigned the last port of the four leading to their subgroup. This leaves four destinations per top-port. Figure 4 shows this for $(2,0,1)$.

Furthermore, each destination has exactly one corresponding source:

$$\begin{aligned}\boldsymbol{C}_{(2,0,1):7}(C2IO(Dmodk)) &= \boldsymbol{C}_{(2,0,1):8}(C2IO(Dmodk)) \\ &= \min(56,4) = 4\end{aligned}$$

$(2,0,1):7$ is the last of $(2,0,1)$'s four ports leading to the left subgroup, and $(2,0,1):8$ is the last leading to the right subgroup.

There are 8 (leaves) times 7 (compute nodes per leaf) $= 56$ compute destinations, to which are assigned all top-ports except for the two ports of $(2,0,1)$ assigned to IO nodes. $= 14$ top-ports, in a balanced manner; this leaves four compute destinations per port. None of these routes are affected by $C2IO$, therefore,

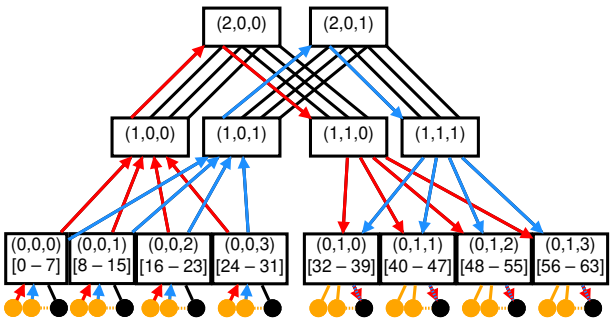


Fig. 5. Two subsets of $C2IO(Smodk)$: red routes have source NID 0 mod 8, light blue routes have source NID 1 mod 8.

$$\forall p \notin (2,0,1), \quad \boldsymbol{C}_p(C2IO(Dmodk)) = 0$$
$$\boldsymbol{C}_{topo}(C2IO(Dmodk)) = 4$$

To reformulate this result: for the given communication pattern most of the top-ports are unused while two top-ports have a strong risk of congestion. This can be seen for one of these ports on Figure 4: the routes shown by the red arrows concentrate around the top-port like those shown in Figure 3.

This is sub-optimal, while spreading both subgroups of four IO destinations any disjoint way among the 8 ports leading to each in the top-switches would have lead to $\boldsymbol{C}_{topo}(C2IO(R_{dst})) = 1$. The object of Section V will be to define such a set of routes R_{dst}.

C. Smodk performance

With Smodk, routes from compute to IO nodes are spread per source. With the same process as Dmodk for compute nodes as destinations, we determine which ports are used with Smodk for computed nodes as sources. More specifically, we will describe which down-port is used as output for switches not directly above the source, and which up-port is used as output for switches directly above, when there are several parallel ports available.

There are 8 top-ports that lead to each subgroup, and 28 compute sources per subgroup; after every group of 7 sources, one NID is skipped, which corresponds to skipping the last considered port of $(2,0,1)$. We conclude that two ports of $(2,0,1)$ have no compute source, and every other top-port has four compute sources which are all connected to different leaves and as a result send to different IO destinations. This results in $\boldsymbol{C}_p(C2IO(Smodk)) = 4$ for each affected top-port p. This is shown in Figure 5.

This means that in this case there are fourteen top-ports with a high risk of congestion; The sets of routes depicted by red and light-blue arrows in Figure 5 also correspond to the situation shown in Figure 3 to show how high congestion risk for two of the fourteen concerned top-ports. For this communication pattern Smodk is less suited than Dmodk.

Optimizing a source-based routing for this pattern and metric means coalescing routes to the same destination. This would be possible for a given pattern; however this article aims to route based on node-type only, therefore we cannot use specific distribution information. If we always have colliding routes lead to distinct destinations, the best we can reach in this situation is still $\boldsymbol{C}_{topo}(C2IO(R_{src})) = 4$.

Since the pattern considered has few destinations and many sources, it is reasonable that a routing algorithm which concentrates routes from the same sources will be difficult to improve. The opposite will happen for the opposite pattern. Section V will also define the set of routes R_{src}.

D. Random routing performance

Dmodk was unable to reach $\boldsymbol{C}_{topo}(Dmodk) = 1$ because the modulo operation depends on NIDs and has no information about the communication pattern. Random routing does not depend on NID; it spreads every route uniformly over the available ports, and as a result every subset of routes is also spread uniformly. Therefore $C2IO(Random)$ does not have particularly coalesced routes.

In practice, distributing each group of 28 routes into its corresponding 8 top-ports always causes collisions between routes that have different destinations. The probability of collision is very close to 1.[2] Therefore, we can safely state that $\boldsymbol{C}_{topo}(C2IO(Random))$ is always greater than 1. Repeated computation of Random routing for the given topology and communication pattern resulted in $\boldsymbol{C}_{topo}(C2IO(Random))$ values of either 3 or 4: i.e. rarely better than Dmodk.

Random routing will usually give slightly better results than Dmodk or Smodk as soon as the communication patterns have a given bias, but they will always leave some ports with avoidable congestion. Just as Xmodk algorithms aim to compute perfect routes for the general worst-case scenario, we want to compute perfect routes for the type-specific worst-case scenario.

V. Grouped Xmodk

In the previous section we show that the existing routing algorithms do not balance the load correctly when the topology has mixed node types.

To improve routing for type-specific communication patterns, we can use knowledge of node types and modify Xmodk algorithms. The aim is to optimize resource usage depending on node type. For example the optimization should achieve the best throughput for communications towards IO proxies or compute nodes.

We suggest balancing each group of nodes separately to improve load-balancing under worst-case type-specific patterns. This corresponds to the previously mentioned R_{dst} and R_{src}.

[2]Determining with what probability there will be a conflict between two of the 28 routes (leading to different destinations) spread through the 8 top-ports is an example of collision probability between sets of random variables [11] (a generalization of the girl/boy birthday problem). However in this case the total number of variables is greater than the number of choices. In that situation, the article's formula for generalized number of sets has a term which always cancels out for part of the computation; it seemed possibly ill-adapted and was therefore discarded.

A. Description of indexing

Grouped Xmodk algorithms, or Gxmodk, consist in preprocessing NIDs. Knowing each node's type, the algorithms begin by updating the NIDs accordingly, as shown in algorithm 1.

Algorithm 1 Reindex NIDs by type

```
from collections import defaultdict
node_types = defaultdict(list)
sort(topo.nodes, key=lambda node: node.NID)
for node in topo.nodes:
    node_types[node.type].append(node)
NID = 0
for node_group in node_types.values():
    for node in node_group:
        node.NID = NID
        NID += 1
```

Note that original order is preserved within groups. Xmodk is then applied as usual but with the updated NIDs.

B. Analysis for previous topology and communication pattern

We choose to call gNID a reindexed NID. Let's suppose that compute nodes are reindexed first: there are 56 so they are assigned gNIDs 0 to 55. IO nodes are assigned gNIDs 56 to 63. Now routing depends on whether Gdmodk or Gsmodk is used.

B.1 Gdmodk results

For Gdmodk, each IO destination is assigned a unique L2 switch in each subgroup (e.g.: gNID 61 is assigned $(1,0,1)$ and $(1,1,1)$). Each L2 switch is assigned two IO destinations of the opposite subgroup, therefore the up-routes from L2 switches use only half of the available parallel ports in a balanced manner.

$$\boldsymbol{C}_{p\in(1,*,*)}(C2IO(Gdmodk)) \leq 1$$

Each L3 switch is shared by two IO destinations for each subgroup (e.g.: $(2,0,1)$ is shared by NIDs 15, 31, 47 and 63; or gNIDs 57, 59, 61 and 63), which are assigned distinct output ports. Figure 6 shows how Gdmok distributes routes efficiently when considering type-based communication patterns. It can be interpreted by pointing out that each set of routes specified by a given color has only one destination (and matches the situation described in Figure 2), with no overlap on output ports in the two upper levels between two sets of routes.

$$\boldsymbol{C}_{p\in(\{1,2\},*,*)}(C2IO(Gdmodk)) = 1$$

All leaves' up-ports have seven sources and two destinations. $\boldsymbol{C}_{p\in Up-ports((0,*,*))}(C2IO(Gdmodk)) = 2$ as is shown with the overlapping dashed red and double-dotted green arrows in Figure 6. It is unavoidable for some of them to have more than one for the given pattern, so Gdmodk gives the best possible quality of routing tables.

$$\boldsymbol{C}_{topo}(C2IO(Gdmodk)) = 2$$

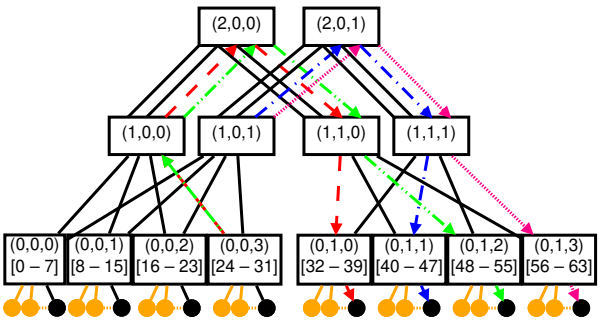


Fig. 6. Simplified representation of all routes going to IO nodes in the right subgroup, under Gdmodk routing. Arrows not displayed can be deduced by shortest paths to displayed ones.

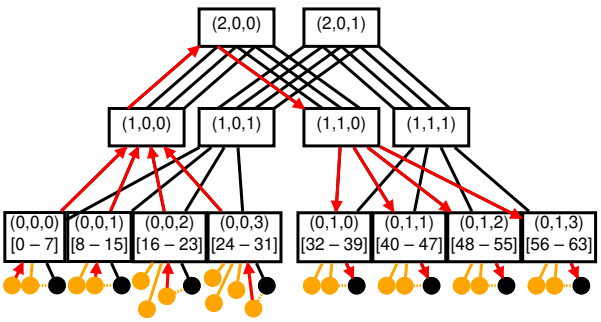


Fig. 7. Subset of $C2IO(Gsmodk)$ with source gNID 0 mod 8

B.2 Gsmodk results

For Gsmodk, the 28 compute sources of each group are assigned all 8 up-ports of the two L2 switches of their subgroup in a balanced manner: there are 7 compute sources per up-port which are used to lead to 4 distinct IO destinations. The 7 top-ports leading to the other subgroup are used the same-way.

$$\boldsymbol{C}_{topo}(C2IO(Gsmodk)) = 4$$

Figure 7 shows all routes of $C2IO(Gsmodk)$ that use one example top-port as output.

Gsmodk improves route distribution for this pattern compared with Smodk: Since an eighth up-port is now used in both L2 switches $(1,*,1)$, (and two down-ports of $(2,0,1)$), each port now has 7 sources. All of these ports that were used by Smodk had 8 sources. This improvement is comparatively minor, because few resources had been spared by Smodk on this pattern. This shows that type-awareness doesn't solve the existing asymmetry issue between optimizing routing to coalesce sources or destinations, but it does improves routing with regards to type-specific patterns any time Xmodk missed out on resources.

On the symmetrical communication pattern we would see the same improvement as we do between Dmodk and Gdmodk for the considered communication pattern. In general, if pattern P is symmetrical to Q, we should always find:

$$\begin{aligned}
\boldsymbol{C}_{topo}(P(Dmodk)) &= \boldsymbol{C}_{topo}(Q(Smodk)) \\
\boldsymbol{C}_{topo}(Q(Dmodk)) &= \boldsymbol{C}_{topo}(P(Smodk)) \\
\boldsymbol{C}_{topo}(P(Gdmodk)) &= \boldsymbol{C}_{topo}(Q(Gsmodk)) \\
\boldsymbol{C}_{topo}(Q(Gdmodk)) &= \boldsymbol{C}_{topo}(P(Gsmodk))
\end{aligned}$$

VI. Conclusions and future works

In this paper we have defined realistic communication patterns depending on node type which are present on our production cluster. From this real-life scenario we analyzed how existing solutions fare against these patterns. We have explicited how type-based communications can result in unnecessary congestion. To counter this we have provided new algorithms to improve existing solutions. We have shown a realistic example with in one case a sevenfold decrease in congestion risk.

The congestion issue of Xmodk stems from nodes of a same type having the same NID, modulo arities. This also affects communications unrelated to node-type, but optimizing for these means knowing about application usage. Gxmodk aims only to improve the situation when node-type is known; having early knowledge of applications' communication matrices would warrant writing specific deterministic algorithms.

This article relies on a static flow metric from which we deduce probable congestion. A more thorough analysis of the relationship between this metric and actual congestion depending on fine-grain communication interaction would also be warranted. A corresponding study of the new algorithms based on simulation rather than only a static congestion metric would also provide results in terms of performance.

This work focuses on fat-trees, for which node indexing allows intuitive understanding of NCA repartition; from this we derive type-based pattern analysis and devise a new node indexing to solve corresponding issues. For other topologies (e.g. DragonFly, Generalized HyperCubes) a similar work could also be attempted. Furthermore, a procedural routing algorithm for fat-trees (which can be useful for routing degraded fat-trees or similar topologies) was omitted; a similar technique could be used to improve it.

Acknowledgements

This BXI development has been undertaken under a cooperation between CEA and Atos. The goal of this cooperation is to co-design extreme computing solutions. Atos thanks CEA for all their inputs that were very valuable for this research.

This research was partly funded by a grant of Programme des Investissments d'Avenir. This work has been jointly supported by the Spanish MINECO and European Commission (FEDER funds) under the project RTI2018-098156-B-C52 (MINECO/FEDER), and by JCCM under project SBPLY/17/180501/000498.

BXI development was also part of ELCI, the French FSN (Fond pour la Société Numérique) cooperative project that associates academic and industrial partners to design and provide software components for new generations of HPC datacenters.

References

[1] Charles E Leiserson, "Fat-trees: universal networks for hardware-efficient supercomputing," *IEEE transactions on Computers*, vol. 100, no. 10, pp. 892–901, 1985.

[2] Fabrizio Petrini and Marco Vanneschi, "k-ary n-trees: High performance networks for massively parallel architectures," in *Proceedings of the 11th International Parallel Processing Symposium*. IEEE, 1997, pp. 87–93.

[3] Sabine R Ohring, Maximilian Ibel, Sajal K Das, and Mohan J Kumar, "On generalized fat trees," in *Proceedings of the 9th International Parallel Processing Symposium*. IEEE, 1995, pp. 37–44.

[4] Eitan Zahavi, "D-mod-k routing providing non-blocking traffic for shift permutations on real life fat trees," *CCIT Report*, vol. 776, 2010.

[5] Jose Rocher-Gonzalez, Jesus Escudero-Sahuquillo, Pedro J García, and Francisco J Quiles, "On the impact of routing algorithms in the effectiveness of queuing schemes in high-performance interconnection networks," in *25th Annual Symposium on High-Performance Interconnects (HOTI)*. IEEE, 2017, pp. 65–72.

[6] Jesus Escudero-Sahuquillo, Ernst Gunnar Gran, Pedro Javier Garcia, Jose Flich, Tor Skeie, Olav Lysne, Francisco Jose Quiles, and Jose Duato, "Combining congested-flow isolation and injection throttling in hpc interconnection networks," in *International Conference on Parallel Processing (ICPP)*. IEEE, 2011, pp. 662–672.

[7] Eitan Zahavi, Gregory Johnson, Darren J Kerbyson, and Michael Lang, "Optimized InfiniBand fat-tree routing for shift all-to-all communication patterns," *Concurrency and Computation: Practice and Experience*, vol. 22, no. 2, pp. 217–231, 2010.

[8] Crispín Gomez, Francisco Gilabert, María Engracia Gomez, Pedro López, and José Duato, "Deterministic versus adaptive routing in fat-trees," in *IEEE International Parallel and Distributed Processing Symposium*. IEEE, 2007, pp. 1–8.

[9] German Rodriguez, Cyriel Minkenberg, Ramon Beivide, Ronald P Luijten, Jesus Labarta, and Mateo Valero, "Oblivious routing schemes in extended generalized fat tree networks," in *IEEE International Conference on Cluster Computing and Workshops*. IEEE, 2009, pp. 1–8.

[10] Pierre Vignéras and Jean-Noël Quintin, "Fault-tolerant routing for exascale supercomputer: The bxi routing architecture," in *International Conference on Cluster Computing (CLUSTER)*. IEEE, 2015, pp. 793–800.

[11] Michael C Wendl, "Collision probability between sets of random variables," *Statistics & probability letters*, vol. 64, no. 3, pp. 249–254, 2003.

Un sistema de mensajería viable para comunidades aisladas basado en LoRa

Miguel Kiyoshy Nakamura Pinto[1], Pietro Manzoni, Carlos Tavares Calafate, Enrique Hernández-Orallo, Juan-Carlos Cano[2]

Resumen— **Más de mil millones de personas en todo el mundo no pueden aprovechar incluso los servicios de conectividad más básicos, la mayoría de los cuales viven en comunidades aisladas. Incluso los servicios más simples de mensajería serían de gran ayuda, por ejemplo, para los agricultores que desean saber el precio de cierto bien que están interesados en vender o comprar antes de decidir si se realiza un viaje posiblemente largo, costoso y agotador.**

En este documento describimos un sistema de bajo coste y bajo consumo basado en el protocolo LoRa para proporcionar un sistema de mensajería sin estar sujeto a costes recurrentes. Las redes LoRa permiten enlaces inalámbricos muy largos que pueden conectar pueblos y ciudades. Además de la aplicación de mensajería simple, LoRa se puede usar para distribuir información de sensores a las comunidades o para proporcionar alertas de desastres. Este sistema se incluye en la categoría de redes comunitarias, donde los usuarios construyen su propia red ya que no hay infraestructura disponible.

Palabras clave— **LoRa; Mensajería; ICT4D; Redes Comunitarias.**

I. Introducción

IN cluso los servicios de mensajería más simples apenas están disponibles en las áreas rurales de los países en desarrollo. La posibilidad de comunicarse entre las aldeas o entre las aldeas y las ciudades principales es un servicio altamente demandado ya sea para uso puramente personal o con fines comerciales. Por ejemplo, los agricultores podrían saber el precio de cierto bien que están interesados en vender o comprar antes de decidir si se emprende un viaje posiblemente largo, caro y agotador.

Las estadísticas de la UIT 2017 informan que las suscripciones de banda ancha móvil han aumentado más del 20% anual en los últimos cinco años y han alcanzado el 56.4% de la población mundial a fines de 2017 [1]. Sin embargo, los precios de banda ancha móvil representan más del 5% del INB (Ingreso Nacional Bruto) per cápita en muchos de los países menos desarrollados y por lo tanto, no son asequibles para la gran mayoría de la población. Mietras que en gran parte de las ciudades disfrutan de conectividad 3G y 4G, la cobertura móvil no existe en muchas áreas rurales. Los que no están cubiertos (brecha de cobertura) son 1,2 mil millones según [2]. La falta de cobertura en las áreas rurales es la consecuencia de un desafío económico básico: la implementación de infraestructura puede ser el doble de cara, mientras que los ingresos pueden ser hasta diez veces más bajos [3]. Donde GSM está presente, SMS ha sido la aplicación más importante para los servicios móviles en todo el mundo debido a su capacidad para mantener la comunicación entre individuos y también para interconectar a los miembros de una comunidad. Es probable que aquellos que se benefician de la cobertura GSM posean un teléfono inteligente: la tasa de penetración es del 44% en todo el mundo y se espera que alcance el 59% para 2022.

DakNet [4] es un ejemplo de un sistema de mensajería destinado a proporcionar servicios más allá de la cobertura de las redes GSM. Aprovecha el transporte existente y la infraestructura de comunicaciones para proporcionar conectividad digital, al combinar un medio físico de transporte con transferencia de datos inalámbrica para ampliar la conectividad a Internet que brinda un cibercafé u oficina postal. DakNet emplea una infraestructura tolerante al retraso para ofrecer aplicaciones como correo de voz, correo electrónico y sistema de boletines electrónicos (BBS) a un costo mucho menor que sus contrapartes en tiempo real. Transmite datos a través de enlaces cortos de punto a punto entre kioscos y un dispositivo portátil a bordo de un vehículo de transporte público (por ejemplo, un autobús) que atiende al área de interés. El bus contiene un simple transceptor WiFi conectado a un servidor y cuando se encuentra dentro del alcance de uno de los dispositivos de comunicación de los kioscos, sincroniza los datos para su procesamiento posterior. El usuario final accede a la información actualizada entregada al kiosco a través de WiFi con su propio teléfono o tableta. Los proyectos piloto han demostrado que las comunicaciones asíncronas pueden satisfacer las necesidades de las personas en áreas remotas [5].

En este trabajo definimos la arquitectura para un sistema de mensajería que combina dispositivos muy baratos, flexibles y la tecnología LoRa para establecer un enlace que puede abarcar áreas amplias con una interfaz fácil de usar. Además, integramos un dispositivo gateway que, si está disponible Internet, puede reenviar mensajes a los usuarios de Telegram.

La solución que desarrollamos tiene como objetivo ofrecer servicios de mensajería a áreas aisladas, no cubiertas por señales GSM. Proporcionar servicios de mensajería de bajo coste y bajo consumo de energía a áreas rurales sin acceso asequible a las redes de comunicaciones existentes, es ciertamente una aplicación importante así como relevante y el sistema descrito que utiliza herramientas de IoT es un enfoque convin-

[1]Departamento de Informática de Sistemas y Computadores, Universitat Politècnica de València, e-mail: minapin@posgrado.upv.es

[2]Departamento de Informática de Sistemas y Computadores, Universitat Politècnica de València, e-mail: {pmanzoni, ehernandez, calafate, jucano}@disca.upv.es

cente para hacerlo. Estamos tratando la mensajería a un nivel alto de abstracción, aprovechando los protocolos y servicios existentes en el mundo de IoT. Además, en caso de desastres, la red LoRa puede utilizarse para intercambiar mensajes de emergencia a largas distancias, incluso en caso de interrupción de la infraestructura celular.

El documento está organizado de la siguiente manera. La Sección 2 describe la tecnología LoRa y sus ventajas, la Sección 3 detalla las motivaciones de esta propuesta. La sección 4 presenta la arquitectura de la solución propuesta. La sección 5 analiza la viabilidad de la solución y la sección 6 presenta las conclusiones.

II. La tecnología LoRa.

Otras soluciones WiFi se han utilizado para enlaces de larga distancia con muy buen rendimiento [6]. Sin embargo, en ciudades de todo el mundo, la proliferación de WiFi ha resultado en serios problemas de interferencia en la banda de 2.4 GHz y aunque menos severa, en la banda de 5.8 GHz y en las dos bandas populares sin licencia. La banda sin licencia de 868 MHz en Europa y 900 MHz en América abre nuevas posibilidades de comunicación, que pueden combatir la interferencia al aprovechar la modulación LoRa [7], en la cual la velocidad de transmisión se negocia por rango, en una aplicación clásica de la fórmula de capacidad del canal de Shannon:

$$C = B \log_2(1 + \frac{S}{N}) \qquad (1)$$

donde C es la capacidad o el rendimiento en b/s, B es el ancho de banda en Hz, S es la potencia de señal en W y N es la potencia de ruido en W.

Es intuitivo que se puede lograr la misma capacidad utilizando un ancho de banda estrecho y una S/N alta o un ancho de banda amplio y una S/N baja. LoRa ofrece un método versátil para reducir el rendimiento para permitir que se produzca la transferencia de datos en instancias de S/N muy bajas, incluso con una potencia de señal del 1 por ciento de la potencia de ruido. Esto se logra controlando la velocidad de datos mediante modificaciones del denominado factor de expansión (SF), que determina el grado de expansión de la velocidad de información (velocidad de bits) a la velocidad de transmisión de datos. El SF varía entre 7 y 12 en Europa, lo que corresponde a velocidades de bits de 10937 b/s a 292 b/s. Cuando el receptor está cerca del transmisor, se puede usar un SF bajo, lo que requiere una sensibilidad del receptor de -123 dBm, mientras que cuando la distancia es mayor o hay obstáculos que reducen la señal recibida, un factor de expansión de 12 decodificará señales tan bajas como -136 dBm, que son mucho más bajas que el ruido térmico (que a temperatura ambiente en un ancho de banda de 125 kHz corresponde también a -123 dBm).

En áreas no cubiertas por proveedores de servicios celulares, a pocos kilómetros de las ciudades y carreteras principales las cuales son muy comunes en países en desarrollo, LoRa puede proporcionar un sistema de comunicación operado por la comunidad y de muy bajo coste que cubre muchas necesidades básicas que abarcan una variedad de casos de uso. En las regiones montañosas, se puede aprovechar la topografía del terreno para lograr transmisiones con línea de visión a distancias muy largas ajustando los nodos con antenas externas de alta ganancia. Una transmisión de 316 km que utiliza nodos de baja potencia y antenas pequeñas se informa en [8] como un ejemplo de lo que se puede lograr. El bajo coste de los dispositivos LoRa, el uso de bandas sin licencia y la amplia disponibilidad de dispositivos habilitados para WiFi (teléfonos inteligentes, tabletas y PC) que proporcionan la plataforma para la interfaz de usuario, constituyen los ingredientes para instalar una red comunitaria, construida y mantenida por los beneficiarios directos, luego de que estén debidamente capacitados.

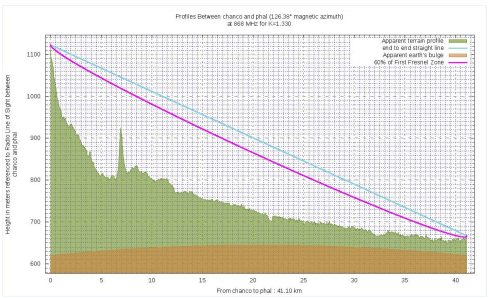


Fig. 1. Perfil del terreno entre el Colegio Canciller de la Universidad de Malawi y el sitio remoto.

Por ejemplo, en 2013 se instaló una red mediante la tecnología *TV white spaces* [9] en el Chancellor College de la Universidad de Malawi para proporcionar acceso a Internet de banda ancha a varias instituciones que lo rodean. Uno de los sitios considerados en la etapa de planificación no se pudo alcanzar con la potencia permitida por la tecnología de *TV White Spaces*. Al usar LoRa, este sitio podría servirse fácilmente ya que hay una línea de visión sin obstrucciones como se muestra en la Figura 1 obtenida con BotRf [10]. El presupuesto de potencia con antenas de 2 dBi en ambos extremos y la potencia del transmisor de 12 dBm calculada con BotRf muestra un margen de 16 dB sobre la sensibilidad de -123 dBm del receptor LoRa con un factor de expansión de 7, como se puede ver en la Figura 2.

III. Motivaciones

En [11] se describe un análisis de las necesidades de comunicación en la atención primaria de salud rural en los países en desarrollo, y aunque muchas aplicaciones interesantes de telemedicina pueden implementarse con soluciones de banda ancha, una aplicación muy simple implementada originalmente sobre la comunicación por radio solo de HF demostró ser bastante exitoso: programar citas con el médico del paciente. Se encontró que los pacientes en áreas aisladas tenían que gastar un tiempo y recursos significativos para llegar al hospital más cercano y con

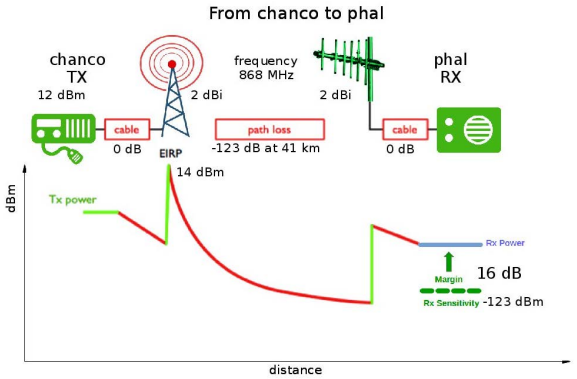


Fig. 2. Potencia sobre distancia entre Chancellor College de la Universidad de Malawi y el sitio remoto utilizando LoRa.

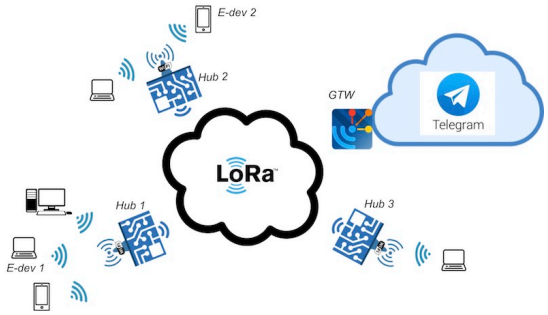


Fig. 3. Estructura general de la plataforma de mensajería.

frecuencia no podían recibir tratamiento de inmediato, sino que se les daba una cita en un momento que a menudo implicaba un segundo viaje desde su casa. Una aplicación de mensajería simple como la que estamos proponiendo, podría servir para organizar una fecha específica en la que el paciente tenga garantizado el tratamiento, ahorrando así tiempo y recursos.

En este trabajo, nos centramos en una aplicación básica del sistema LoRa, que es una mensajería individualizada de tipo SMS. Aunque esto es muy útil cuando no hay otros sistemas implementados, se pueden desarrollar más aplicaciones utilizando la misma solución. Por ejemplo, se puede proporcionar una funcionalidad similar al sistema de boletín de anuncios (BBS) a los habitantes de áreas aisladas. Se puede utilizar para ofrecer datos de pronóstico del tiempo, información de atención médica, precios de mercado de los cultivos, etc. En este caso, la comunicación es de uno a muchos, ya que todos pueden leer los mensajes.

Finalmente, nuestra solución propuesta también puede aprovecharse para admitir aplicaciones LoRa normales, como sensores ambientales, pronóstico del tiempo, etc. Si estos sensores están equipados con transceptores LoRa, pueden ubicarse fuera del alcance de las torres de telefonía celular y enviar datos al *hub* LoRa, que puede enviar los datos a las partes interesadas. En particular, los mensajes para mitigaciń de desastres utilizando las comunicaciones mediante dispositivos LoRa pueden ser muy valiosas cuando se interrumpe el servicio celular y también para extender la conectividad más allá del rango de las torres celulares.

IV. La arquitectura propuesta.

En el núcleo de la arquitectura propuesta se encuentran dispositivos dedicados, llamados *hubs*, que crean el punto de conectividad dentro de un área. Los *hubs* deben tener WiFi (IEEE802.1b/g/n) y un transceptor LoRa. La figura 3 presenta la arquitectura general.

Los *hubs* funcionan como punto de acceso WiFi estándar para proporcionar conectividad a dispositivos cercanos. La interfaz con la aplicación de mensajería es una página basada en web (vea la figura 4). El usuario puede decidir si enviar un mensaje de texto a un destino específico o verificar los mensajes entrantes almacenados en el *hub*. Cada usuario debe "registrarse" antes de intercambiar cualquier mensaje. Es necesario registrarse para permitir que el sistema localice el destino final, que posiblemente pueda moverse a diferentes ubicaciones en el tiempo.

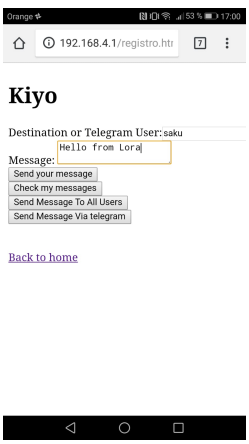


Fig. 4. Captura de pantalla de la interfaz basada en web.

Los *hubs* reciben comandos POST de los dispositivos conectados para enviar un mensaje o devolver los recibidos previamente y los almacenados localmente.

Cuando un usuario envía un mensaje, el *hub* local "aprende" que ese usuario está conectado a través de él y crea un registro en una tabla. El primer paso es descubrir dónde se encuentra el usuario de destino. Para ello, el *hub* envía un mensaje *broadcast* utilizando la capa física del protocolo LoRa a todos los dispositivos circundantes. El dispositivo que tiene ese usuario final registrado, responde al *hub* solicitante. La estructura de paquetes utilizada por este protocolo es muy simple, con dos bits para control y un campo de 32 bytes para almacenar el nombre que se va a buscar. Se incluyó un usuario *broadcast* para los mensajes que se entregarán a todos los usuarios registrados.

La figura 5 muestra el conjunto ordenado de resultados del proceso de descubrimiento del usuario. El valor mediano para todo el proceso es 2.077 mseg. Una vez que se conoce la ubicación del usuario, este

proceso no se repite.

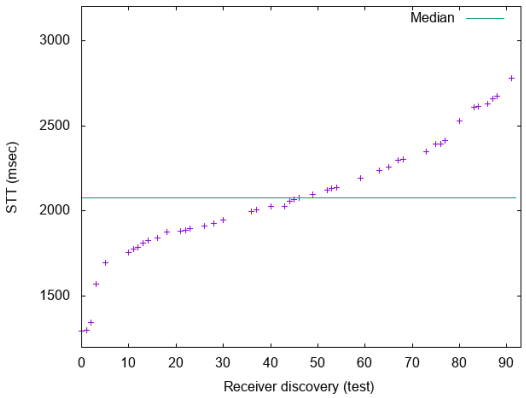


Fig. 5. Proceso de descubrimiento del usuario; conjunto ordenado de resultados ("tests").

Luego, utilizando un protocolo confiable de unidifusión, el mensaje se transfiere y almacena en el *hub* de destino. Una vez que el usuario a quien se dirige el mensaje verifica el mensaje disponible, recibirá el que está almacenado en el *hub* local. El protocolo de unidifusión se basa en un enfoque ARQ clásico de parada y espera con un valor dinámico y adaptable para el retardo de retransmisión. El protocolo garantiza que la información no se pierda debido a paquetes caídos y que los paquetes se reciban en el orden correcto. La estructura del paquete se muestra en la Figura 6. Debemos señalar que la carga útil máxima de la aplicación depende de la velocidad de datos seleccionada. Por ejemplo, suponiendo en la banda europea 863-870 MHz, para que un nodo funcione en las peores condiciones de propagación, se debe asumir la velocidad de datos más baja, proporcionada por SF12, donde el nodo no puede enviar más de aproximadamente 51 bytes por paquete; con SF7, la carga útil podría ser de 222 bytes. Estos valores son considerados en el protocolo LoRaWAN que agrega al menos 13 bytes a la carga útil de la aplicación. El tamaño máximo de paquete utilizado se estableció de acuerdo con el factor de propagación utilizado, es decir, 25 bytes para SF12 y 200 bytes para SF7, con un encabezado fijo de 24 bytes.

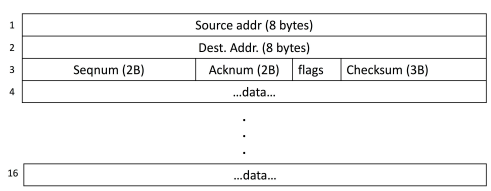


Fig. 6. Estructura del paquete utilizado por el protocolo ARQ de stop-and-wait.

A. *Integración con Telegram.*

Para integrar mejor nuestra arquitectura con la aplicación de Internet estándar, diseñamos un *hub gateway* para vincularlo con Telegram [1], una aplicación de mensajería ampliamente utilizada. Seleccionamos Telegram ya que ofrece los llamados *Bots*. Los *bots* son aplicaciones de terceros que se ejecutan dentro de Telegram. Los usuarios pueden interactuar con los *bots* enviándoles mensajes, comandos y solicitudes en línea.

El *gateway* recibe a través de mensajes LoRa dirigidos a un usuario de Telegram, registrado a través del *bot*, y los envía al teléfono del usuario a través del enlace de Internet.

El *gateway* de nuestro prototipo se basa en una placa Raspberry Pi provista de un adaptador Lora y una conexión estable a Internet. Los procesos requeridos por el *bot* se están ejecutando en la Raspberry Pi.

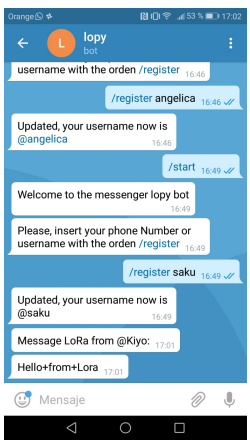


Fig. 7. Ejemplo de interacción con el Bot Telegram.

B. *Algunos resultados preliminares.*

En nuestros experimentos, utilizamos dispositivos llamados LoPy [2], un microcontrolador habilitado para MicroPython, basado en el último conjunto de chips ESPressif ESP32. Tiene un procesador dual y tres transceptores (LoRa, WiFi, Bluetooth) y dos conectores de antena: uno para la banda de 868 MHz utilizada por LoRA y otro para la banda de 2.4 GHz utilizada por WiFi y Bluetooth. Esto permite instalar la antena más adecuada para la aplicación. Por ejemplo, se pueden usar antenas direccionales de alta ganancia para conectar una aldea rural a una ciudad que podría estar a una distancia muy larga, usando una potencia muy pequeña al aprovechar las características de *spread spectrum* de la modulación LoRa. El procesador de red maneja la conectividad WiFi y la pila de IP, mientras que el procesador principal es totalmente libre de ejecutar la aplicación de usuario. Tiene una memoria RAM de 512 KB y un flash externa de 4 MB. Tiene una aceleración de punto flotante de hardware y se puede programar en multihilo de Python.

El *hub* completo se muestra en la Figura 8: un nodo LoPy con una antena omnidireccional, una batería externa y un panel solar de 5W.

Las siguientes gráficas se basaron en un parámetro llamado " tiempo de transferencia exitoso (STT)"

[1] https://telegram.org/

[2] https://pycom.io/

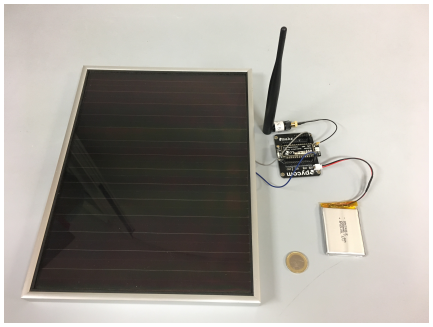


Fig. 8. Sistema general: LoPy, batería y panel solar.

que mide el tiempo de transferencia de un mensaje desde el punto de vista del remitente. Se calcula desde el momento en que se envía el primer fragmento del mensaje hasta el momento en que se recibe el último ACK del último fragmento del mensaje.

La figura 9 muestra el comportamiento del STT al variar la distancia entre dos nodos. Se muestran los valores medianos. Los gráficos se obtienen utilizando un factor de dispersión (SF) de 7 y un tamaño de mensaje de 1, 256 y 512 bytes. Como puede verse, hay un comportamiento estable en los resultados, aunque claramente el STT crece a medida que aumenta el tamaño del mensaje; en SF7 el tamaño máximo para la carga útil era de 200 bytes.

Parece que el sistema es bastante estable para el aumento de la distancia y muy pocas retransmisiones fueron necesarias durante el experimento. En general, tuvimos una tasa de retransmisión del 9%, pero debemos señalar que la distribuciń de las retransmisiones fue prácticamente independientes de la distancia.

Los valores absolutos del STT resaltan que el tiempo de configuración requerido por los dos dispositivos para sincronizar el transmisor tiene la mayor sobrecarga, seguido del envío de datos reales.

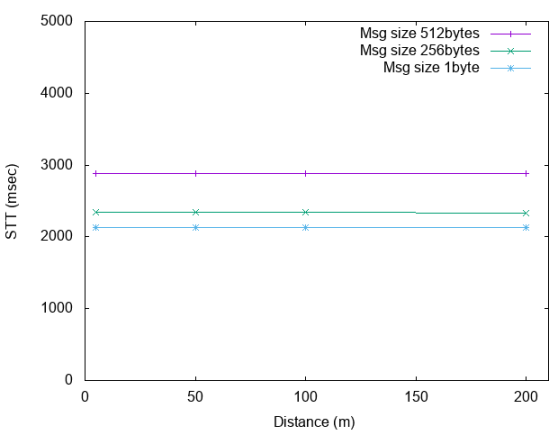


Fig. 9. El comportamiento del STT al variar la distancia entre dos nodos (valores medianos).

La figura 10 permite ver mejor la evolución de la STT en función del tamaño del mensaje. Como podemos ver, el STT crece claramente a medida que aumenta el tamaño del mensaje, pero se evidencia que el impacto de la distancia es insignificante.

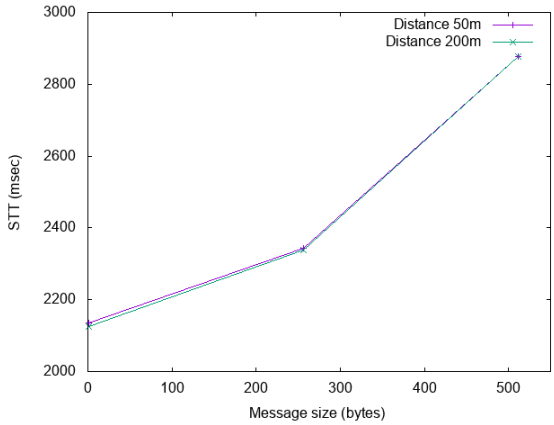


Fig. 10. El comportamiento de la STT al variar el tamaño del mensaje (valores medianos).

Finalmente, la Figura 11 muestra el comportamiento de la STT cuando se varía la frecuencia de los mensajes generados, más exactamente, cuando aumenta la cantidad de mensajes simultáneos generados. Los gráficos se obtienen utilizando un factor de propagación 7 y un tamaño de mensaje de 256 bytes. Como puede verse, el STT crece linealmente a medida que aumenta la cantidad de mensajes simultáneos, lo que da lugar a un caso extremo en el que 14 usuarios simultáneos intentaban enviar un mensaje; en unos pocos casos se obtuvo un retraso máximo de 80 segundos. El valor medio, para los 14 usuarios simultáneos, fue de 35.892 seg. Con más de 14 usuarios simultáneos, el sistema mostró un comportamiento muy poco confiable y lo consideramos básicamente inutilizable.

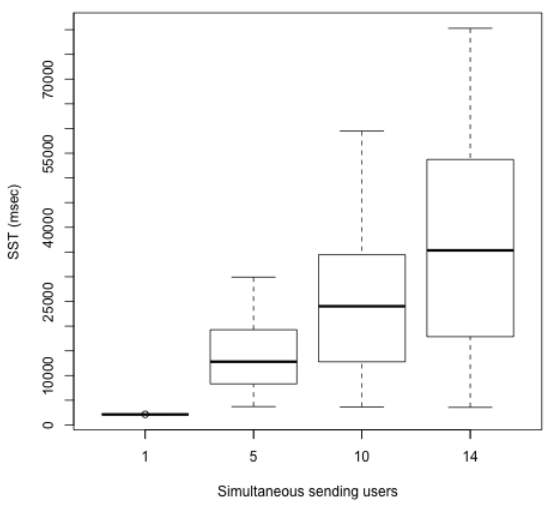


Fig. 11. Evaluación de la escalabilidad del sistema al aumentar el número de usuarios que envían simultáneamente.

V. Viabilidad de la solución.

Para que una solución de red sea viable, se deben tener en cuenta varios aspectos. A continuación se presenta el análisis de viabilidad de la solución prop-

uesta.

A. Solución independiente de red eléctrica.

Medimos el consumo de energía de la solución propuesta en dos condiciones: comunicación solo WiFi y comunicación WiFi+LoRa. En el primer caso (al compilar el mensaje en la interfaz web) medimos 0.476 W, en el segundo caso (al enviar un mensaje a través de LoRa) medimos 0.587 W con un aumento del 23% en el consumo de energía. Luego comparamos nuestra solución basada en dispositivos LoPy con una plataforma alternativa basada en la placa Raspberry Pi y Uputronics LoRa, y medimos un consumo de energía de 1.454 W, un aumento del consumo de 248% en comparación con nuestra solución. Dado que el WiFi siempre debe estar encendido para atender las solicitudes entrantes de los usuarios, el *hub* no puede aprovechar el modo de suspensión profunda, pero se investigarán otras funciones de ahorro de energía en trabajos futuros.

Con un consumo de energía máximo de 0.587 W, el dispositivo se puede alimentar fácilmente a través de fuentes renovables como la solar. Una batería de 3.7 V, 2000 mAh puede alimentar el *hub* durante 17 horas y un panel solar de 5 W es suficiente para reponer la batería en la mayoría de los climas.

B. Viabilidad económica.

Las arquitecturas de red tradicionales se basan en el supuesto de que existe una infraestructura fija y que los usuarios finales hacen uso de esta infraestructura mediante el pago de una tarifa (este es el caso de redes GSM, satelitales y WiFi). Si bien este modelo funciona bien en los países industrializados (donde las empresas o el gobierno se encargan de la inversión inicial), constituye una gran barrera en muchos países en desarrollo, especialmente en áreas rurales y remotas. La solución propuesta requiere una inversión inicial limitada que se puede compartir entre los miembros de la comunidad y no requiere ningún costo recurrente. Este es el mismo modelo que las redes comunitarias, donde los enlaces WiFi se configuran en una comunidad y no estÃ¡n necesariamente conectados a Internet. Otro ejemplo es la telefonía celular comunitaria, donde las comunidades en áreas rurales instalan su propia red GSM [12]. La arquitectura de red propuesta se basa en dispositivos LoRa de bajo costo que cuestan alrededor de 30 euros. El costo total del dispositivo, incluido el panel solar, la baterÃa y el gabinete, es de aproximadamente 70 euros.

C. Regulación.

Los dispositivos LoRa operan en frecuencias de 868 MHz en Europa con una potencia de salida mÃ¡xima de 14 dBm y 915 MHz en América del Norte y del Sur, Australia y Nueva Zelanda con una potencia máxima de 20 dBm. Estas frecuencias forman parte de las bandas ISM que no requieren ninguna licencia para operar. Mientras que las frecuencias son utilizables, la regulación está en su lugar para permitir el uso justo del espectro al dictar el ciclo útil de los dispositivos. Un 1% de ciclo útil significa que el transmisor solo puede transmitir el 1% del tiempo. La mayoría de los países europeos siguen la Recomendación 70-03 de la CEPT/ERC relacionada con el uso de dispositivos de corto alcance (SRD), mientras que una docena de países africanos se rigen por la Asociacion Reguladora de Comunicaciones del sur de África (CRASA) para aplicaciones SRD en bandas de frecuencia armonizadas.

VI. Conclusiones.

Hemos presentado el diseño de una solución de bajo consumo y bajo coste para proporcionar un sistema de mensajeria a comunidades aisladas. Consideramos necesaria una solución como la nuestra, ya que todavía hay una población importante que puede beneficiarse de ella en las zonas rurales de los países en desarrollo. Una vez en su lugar, el *hub* LoRa también se puede utilizar para sensores y otros tipos de datos de interés.

Nuestra plataforma integra un *gateway* a Telegram, una aplicación de mensajería ampliamente adoptada. Este dispositivo permite extender el alcance del sistema de mensajería a Internet estándar.

Desarrollamos un prototipo para obtener algunos resultados de su desempeño. Los resultados obtenidos muestran que esta propuesta puede ofrecer una solución eficiente y de bajo coste para el contexto al que nos dirigimos. Además, como se indica en el texto, esta arquitectura se puede ampliar y utilizar para otras aplicaciones como ofrecer datos de pronóstico del tiempo, información de atención médica, precios de los mercados de cultivos, etc.

Se planean muchas mejoras para esta plataforma que muestra amplias posibilidades. Actualmente estamos trabajando en mensajes de voz para facilitar el uso de la plataforma a los usuarios que no se sienten cómodos leyendo y escribiendo. También exploraremos la posibilidad de instalar en un autobús u otro vehículo de transporte público que atraviese áreas no atendidas por el operador de telefonía celular con un *hub* LoRa. Esto le permitirá recopilar mensajes de texto LoRa que luego pueden reenviarse al destino deseado una vez que el vehículo alcance la cobertura del servicio celular. Otra evolución importante está orientada a incluir un *gateway* a SMS con posiblemente la integración de plataformas de pago como M-PESA [13]. En este último caso, se considerará la cuestión de seguridad.

Agradecimientos

Este trabajo fue parcialmente apoyado por el "Ministerio de Ciencia, Innovación y Universidades, Programa Estatal de Investigación, Desarrollo e Innovación en Orientación a los Retos de la Sociedad, Proyectos I+D+I 2018", España, bajo la subvención RTI2018-096384-B-I00.

Referencias

[1] ITU, "Global ict developments, 2001-2017," Tech. Rep., ITU, 02 2018.

[2] Genaro Cruz and Guillaume Touchard, "Enabling rural coverage: regulatory and policy recommendations to foster mobile broadband coverage in developing countries," Tech. Rep., GSMA, 02 2018.

[3] J. Simo-Reigadas, E. Municio, E. Morgado, E. M. Castro, and A. Martinez, "Sharing low-cost wireless infrastructures with telecommunications operators for backhauling 3g services in deprived rural areas," in *2015 IEEE 16th International Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, June 2015, pp. 1–8.

[4] A. Hasson A. Pentland, R. Fletcher, "Daknet:rethinking connectivity in developing nations," *Computer*, vol. 37, pp. 78–83, 2004.

[5] Adriano Galati, Theodoros Bourchas, Sandra Siby, and Stefan Mangold, "System architecture for delay tolerant media distribution for rural south africa," in *Proceedings of the 9th ACM International Workshop on Wireless Network Testbeds, Experimental Evaluation and Characterization*, New York, NY, USA, 2014, WiNTECH '14, pp. 65–72, ACM.

[6] Rob Flickenger, Steve Okay, Ermanno Pietrosemoli, Marco Zennaro, and Carlo Fonda, "Very long distance wi-fi networks," in *Proceedings of the second ACM SIGCOMM workshop on Networked systems for developing regions*. ACM, 2008, pp. 1–6.

[7] Lorenzo Vangelista, Andrea Zanella, and Michele Zorzi, "Long-range iot technologies: The dawn of lora," in *Future Access Enablers of Ubiquitous and Intelligent Infrastructures*. Springer, 2015, pp. 51–58.

[8] Nikola Jovalekic, Ermanno Pietrosemoli, Marco Rainone, and Marco Zennaro, "Smart and very distant objects," in *Proceedings of the 3rd Workshop on Experiences with the Design and Implementation of Smart Objects*. ACM, 2017, pp. 29–34.

[9] Chomora Mikeka, Justice Stanley Mlatho, Martin Thodi, Jonathan Pinifolo, Dereck Kondwani, Lloyd Momba, Marco Zennaro, Andrés Arcia-Moret, Carlo Fonda, and Ermanno Pietrosemoli, "Preliminary performance assessment of tv white spaces technology for broadband communication in malawi," *Procedia Engineering*, vol. 78, pp. 149–154, 2014.

[10] Marco Zennaro, Marco Rainone, and Ermanno Pietrosemoli, "Radio link planning made easy with a telegram bot," in *International Conference on Smart Objects and Technologies for Social Good*. Springer, 2016, pp. 295–304.

[11] Andrés Martinez, Valentín Villarroel, Joaquín Seoane, and Francisco del Pozo, "Analysis of information and communication needs in rural primary health care in developing countries," *IEEE transactions on Information Technology in Biomedicine*, vol. 9, no. 1, pp. 66–72, 2005.

[12] Aditya Dhananjay, Matt Tierney, Jinyang Li, and Lakshminarayanan Subramanian, "Wire: A new rural connectivity paradigm," *SIGCOMM Comput. Commun. Rev.*, vol. 41, no. 4, pp. 462–463, Aug. 2011.

[13] Tavneet Suri William Jack, "Mobile Money: The Economics of M-PESA," Tech. Rep., National Bureau Of Economic Research - Working Paper No. 16721, 2011.

[14] Marco Zennaro, Carlo Fonda, Ermanno Pietrosemoli, A Muyepa, Steve Okay, Rob Flickenger, and SM Radicella, "On a long wireless link for rural telemedicine in malawi," 2008.

Mejora de la Transmisión de Vídeo en Redes Vehiculares mediante Calidad de Servicio

P. Pablo Garrido Abenza, Pablo Piñol Peral, Manuel P. Malumbres, O. López Granado[1]

Resumen— **La transmisión de vídeo en redes vehiculares puede tener un gran número de aplicaciones. Sin embargo, éstas tienen unas propiedades que las diferencian de las redes cableadas y que dificultan la transmisión de contenido multimedia. El canal inalámbrico es un medio compartido, que tiene un ancho de banda limitado y una naturaleza cambiante a lo largo del tiempo. A todo ello se suma que la transmisión de vídeo requiere de un gran ancho de banda, y que la topología de la red cambia con mucha frecuencia debido a la velocidad a la que se mueven los vehículos. En este trabajo se realizan diversos experimentos mediante simulación utilizando diversas técnicas existentes con objeto de mejorar la transmisión de vídeo en este tipo de redes. Con un entorno de simulación basado en OMNeT++, Veins y SUMO, se analizan algunos modos de codificación con refresco de frames de tipo I, particionado de frames (*tiles*), y calidad de servicio mediante la asignación de prioridad dependiendo del tipo de frame. Los resultados de estos experimentos muestran que mediante la combinación de estas técnicas se consigue que la transmisión de vídeo en redes vehiculares sea más robusta.**

Palabras clave— **Redes vehiculares, Vídeo, HEVC, QoS**

I. Introducción

Las Redes Vehiculares o Vehicular Ad-hoc Networks (VANETs) tienen un gran potencial puesto que contribuyen a los llamados Intelligent Transportation Systems (ITS), proporcionando una serie de servicios en entornos urbanos e inter-urbanos, para conductores y pasajeros. Entre ellos, podemos mencionar la implementación de sistemas relacionados con la mejora de la seguridad en la conducción, de información de la situación del tráfico o previsión meteorológica, acceso a Internet, y aplicaciones de entretenimiento (*infotainment*).

Sin embargo, las comunicaciones en este tipo de redes tiene una serie de problemas. Por un lado, tenemos los problemas típicos de las comunicaciones inalámbricas, como un ancho de banda limitado, que al ser compartido por todos los dispositivos de la red genera interferencias con las señales de otros dispositivos, o de otras redes inalámbricas que se solapan en su zona de cobertura. Además, se producen diversos fenómenos como la atenuación de la señal con la distancia (*path loss*) o el tiempo (*fading*), la presencia de obstáculos (*shadowing*), y los rebotes debidos a refracción o reflexión (*multipath*). Por otro lado, debido a la movilidad inherente de los nodos de la red, la topología de la red es cambiante, y la alta velocidad de los vehículos limita el tiempo de comunicación. Todo ello se traduce en un aumento del tiempo de espera para el acceso al canal, y un aumento también del número de paquetes perdidos debido a colisiones, reduciendo su rendimiento en comparación con las redes cableadas [1].

La transmisión de vídeo en redes vehiculares puede tener muchas aplicaciones, como la difusión de anuncios o información turística según nuestra posición, transmisión de vídeo en tiempo real o *video streaming*, videovigilancia, visualización del estado del tráfico en alguna zona, etc. Sin embargo, transmitir vídeo con calidad suficiente es muy complicado, puesto que requiere un gran ancho de banda, bajo retardo o latencia (*delay*) y una variación de retardo (*jitter*) acotada. Con el uso de técnicas de codificación o compresión de vídeo se reduce la cantidad de datos necesaria para su almacenamiento, así como el ancho de banda requerido para la transmisión. En los últimos años han surgido diversos estándares de codificación de vídeo, como High Efficiency Video Coding (HEVC) [2], que mejora las tasas de compresión de su predecesor H.264/AVC (Advanced Video Coding) [3]. Aun así, la calidad de vídeo percibida por el receptor puede verse muy afectada por los problemas ocurridos durante la transmisión, en particular, en escenarios de redes vehiculares. Para tratar de solucionar los problemas en la transmisión de vídeo en redes vehiculares, se suele recurrir a técnicas y mecanismos de control, recuperación y ocultación de errores que tratan de maximizar la calidad del vídeo percibida por el usuario. Estas técnicas, permiten dotar al servicio de transmisión de vídeo de un nivel determinado de calidad de servicio, Quality of Service (QoS). Diversos trabajos como [4] y [5] agrupan estas técnicas en varias categorías: (1) control de admisión y reserva de ancho de banda, (2) QoS a nivel aplicación, (3) diferenciación de tráfico a nivel Medium Access Control (MAC), y (4) adaptación del enlace en la capa física (PHY). Las técnicas de control de errores suelen actuar a nivel de aplicación. En caso de utilizar protocolos de comunicación asíncrona como Automatic Repeat Request (ARQ), el receptor debe enviar un acuse de recibo (ACK) en caso de haber recibido correctamente un paquete, y el emisor retransmitirá aquellos paquetes sin acuse de recibo. Sin embargo, esto no es válido durante el envío de una secuencia de vídeo en tiempo real, en la que se requiere una baja latencia. Otra alternativa son los protocolos de comunicación síncronos, como el uso de Forward Error Correction (FEC), que permiten reconstruir los datos perdidos a partir de la información redundante enviada en otros paquetes, siempre y cuando el número de paquetes perdidos no supere un determinado límite. Estos mecanismos no son adaptativos, por lo que se puede desperdiciar

[1]Dpto. Ingeniería de Computadores, Universidad Miguel Hernández, Elche, e-mail: pgarrido, pablop, mels, otoniel@umh.es.

ancho de banda por los paquetes añadidos innecesariamente en caso de que la red no esté saturada, o, por el contrario, en caso de estar saturada podría no ser suficiente para poder restaurar todos los paquetes perdidos.

En cuanto al uso de QoS mediante diferenciación de tráfico a nivel MAC, éste puede realizarse de dos modos: (1) mediante planificación de colas, o (2) mediante la asignación de diferentes niveles de prioridad. Para el caso particular de la transmisión de secuencias de vídeo, este trabajo se centra en priorizar los paquetes en función del tipo de frame al que pertenecen (I, P, B). Se han planificado dos conjuntos de experimentos mediante simulación: (1) priorizando sólo los paquetes pertenecientes a los frames de tipo I, y (2) priorizando todos los paquetes de vídeo (frames I, P, B) respecto al resto de tráfico.

A la vista de los resultados, los modos AI o LPI4, con 6 u 8 tiles por frame, y priorizando todos los paquetes de vídeo (recomendación del estándar) parece ser la alternativa que proporciona los mejores resultados. Utilizando diferenciación de tráfico se obtiene una mejor calidad de vídeo que con el servicio *Best-effort* proporcionado por defecto del IEEE 802.11.

La estructura del artículo es la siguiente. Primero, en la sección II se hace un breve repaso de los estándares de comunicación utilizados en entornos vehiculares, como son el IEEE 802.11, y el IEEE 1609.4. A continuación, en la sección III se presentan algunas trabajos existentes en la literatura relacionados con el uso de QoS en la transmisión de contenido multimedia en redes inalámbricas. En la sección IV se describe en detalle el escenario utilizado para los experimentos, así como la secuencia de vídeo transmitida y los modos de codificación HEVC utilizados. Los resultados de los experimentos se discuten en la sección V. Por último, la sección VI finaliza el artículo y enumera algunos de los trabajos futuros.

II. Estándares de comunicación

En el estándar IEEE 802.11 [6] se define una subcapa Medium Access Control (MAC), así como varias capas a nivel físico (PHY). A pesar de que el IEEE 802.11 es el tipo de red inalámbrica más extendido, no incluye soporte para QoS.

El grupo de trabajo IEEE 802.11e [7] definió algunas extensiones al estándar IEEE 802.11 para proporcionar QoS a nivel de la capa MAC. Habilitando diferenciación de tráfico a nivel MAC es posible dar soporte a tráfico de distintas aplicaciones dependiendo de sus restricciones de QoS, como pueden ser las llamadas Voice over IP (VoIP), la videoconferencia, la videovigilancia, y cualquier otra aplicación en tiempo real. El estándar IEEE 802.11e introdujo la Hybrid Coordination Function (HCF), la cual define dos nuevos mecanismos de acceso que sustituyen a Distributed Coordination Function (DCF) y Point Coordination Function (PCF) del IEEE 802.11: el HCF Controlled Channel Access (HCCA) y el Enhanced Distributed Channel Access (EDCA). Este último, el EDCA, es el que deben implementar los nodos de una red inalámbrica Ad hoc (sin infraestructura). A los paquetes a transmitir se les asigna una prioridad denominada User Priority (UP) a nivel aplicación, cuyo valor está comprendido entre 0 (la menor) y 7 (la mayor). Cuando la capa MAC recibe un paquete de las capas superiores, éste se clasifica según una de las cuatro categorías de acceso, o Access Categories (AC), es decir, que a cada AC se le asignan dos UP, tal como se muestra en la Tabla I. La categoría AC(3) es la de mayor prioridad, y la AC(0) la de menor prioridad.

TABLA I: Correspondencia entre UP y AC.

User Priority (UP)	Access Category (AC)	Nombre
1	AC_BK o AC(0)	Background
2	AC_BK o AC(0)	Background
0	AC_BE o AC(1)	Best Effort
3	AC_BE o AC(1)	Best Effort
4	AC_VI o AC(2)	Video
5	AC_VI o AC(2)	Video
6	AC_VO o AC(3)	Voice
7	AC_VO o AC(3)	Voice

Los nodos que implementan EDCA tienen 4 colas de servicio en la capa MAC (una para cada AC), con objeto de clasificar los paquetes de acuerdo a su prioridad. Si existen paquetes a transmitir en varias de las colas se produce una colisión interna (virtual), seleccionándose la cola de mayor prioridad. Además, la capa MAC también tratará de forma diferente a los paquetes a la hora de acceder al medio dependiendo de la prioridad asignada, en función de los límites inferior y superior de la ventana de contención o Contention Window (CW_{min} y CW_{max}), el Arbitration Inter-Frame Space Number (AIFSN), y la Transmission Opportunity (TXOP).

El parámetro AIFSN es un tiempo adicional que se debe esperar antes de transmitir una vez que se detecte que el medio está libre, y es específico para cada una de las colas de las ACs. En caso de que el canal esté ocupado, el dispositivo tendrá que iniciar un proceso de *backoff*, en el que tendrá que esperar un tiempo proporcional a un valor aleatorio comprendido en el rango $[0..CW]$, donde el valor inicial de la ventana de contención es CW_{min}. Si pasado ese tiempo el medio sigue ocupado, la ventana de contención se incrementará de forma exponencial, hasta llegar al valor máximo CW_{max}. Las colas de mayor prioridad tienen valores menores de CW y AIFSN para acceder al canal. Por último, el parámetro TXOP define un intervalo de tiempo en el que el nodo puede transmitir sin que exista competencia con otros nodos. Esto permite obtener un mayor rendimiento, que será mayor para la cola que tenga un mayor valor de TXOP, así como un incremento en la ocupación global del canal.

Sin embargo, las redes vehiculares, además de las características propias de las redes inalámbricas, tienen el problema de la alta movilidad de los vehículos, que provoca que la topología de la red cambie frecuentemente y las comunicaciones no puedan du-

TABLA II: Valores por defecto de los parámetros EDCA para IEEE 802.11p.

AC	$CW_{min..max}$	AIFSN	$TXOP_{limit}$
AC_BK	15..1023	9	0 ms
AC_BE	15..1023	6	0 ms
AC_VI	7..15	3	0 ms
AC_VO	3..7	2	0 ms

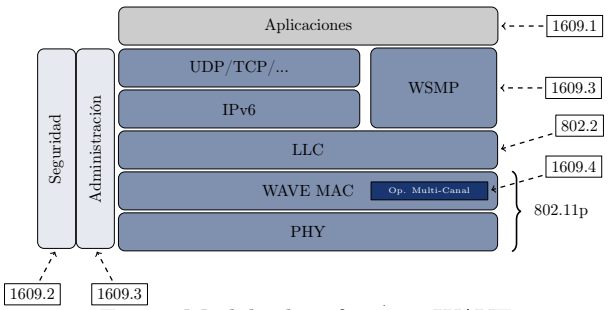


Fig. 1: Modelo de referencia WAVE

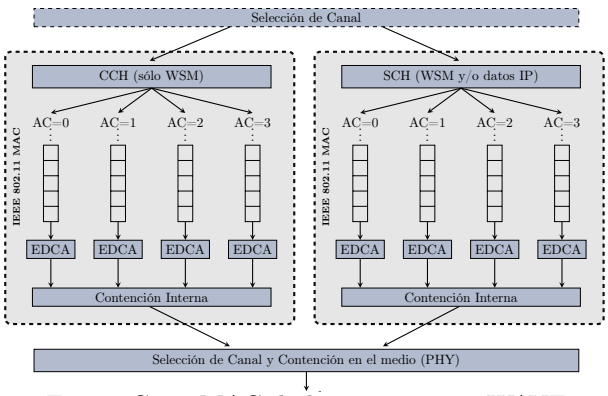


Fig. 2: Capa MAC de la arquitectura WAVE

rar mucho tiempo. Por ello, se propuso el conjunto de estándares IEEE 1609, conocido como Wireless Access in Vehicular Environments (WAVE), que proporciona una pila de protocolos de comunicación optimizada para entornos vehiculares (Fig. 1). En concreto, el IEEE 1609.4 [8] especifica las extensiones que es necesario realizar a la capa MAC del IEEE 802.11, para la necesaria coordinación entre el uso de un canal de control, Control CHannel (CCH) y un canal de servicio, Service CHannel (SCH), incluyendo una capa MAC para cada uno de ellos (Fig. 2). Las características de esta capa MAC específicamente diseñada para entornos de redes vehiculares se definen en el estándar IEEE 802.11p [9], que está basado en el IEEE 802.11e con ciertas modificaciones. En concreto, modifica ligeramente los valores por defecto de los parámetros EDCA de la subcapa MAC (Tabla II). La capa física (PHY) del IEEE 802.11p es similar a la del estándar IEEE 802.11a, pero soporta como máximo una tasa de transmisión de hasta 27 Mbps, es decir, la mitad que aquel.

WAVE soporta tanto transferencias de datos IP como no-IP. Las transferencias no-IP consisten en el envío de mensajes cortos o WAVE Short Messages (WSMs), definidos en el estándar IEEE 1609.3 o WAVE Short Message Protocol (WSMP). Otros estándares incluidos en la arquitectura WAVE son el IEEE 1609.2, que especifica los servicios de seguridad, y el IEEE 1609.0, que describe la arquitectura y operaciones de WAVE (Fig. 1).

III. Trabajos relacionados

Existen distintas propuestas existentes en la literatura que asignan distinta prioridad en función del tipo de frame, bien de forma estática [10] [11] [12], o bien, de forma dinámica como el Dynamic Frame Assignment Algorithm (DFAA) [13] y otros [14] [15].

Por ejemplo, en [10] se realiza una partición de datos a nivel aplicación (*slices*), a los cuales se les asigna una prioridad (AC) a nivel MAC IEEE 802.11e en función del tipo de partición, de forma estática y utilizando todas las AC.

En [16] se propone un mecanismo adaptativo denominado Adaptive Mapping Mechanism (AMM) para mejorar la calidad de vídeo H.264 transmitido sobre redes inalámbricas (WLAN) basadas en IEEE 802.11e, mediante la asignación de distinta prioridad en función de la estructura del vídeo codificado (tipo de frame), importancia del frame, y la carga de cada Access Category (AC). El mecanismo AMM se compara con el mecanismo EDCA del IEEE 802.11e, así como con otras propuestas estáticas y dinámicas.

Sin embargo, los trabajos anteriores no están orientados a redes vehiculares (IEEE 802.11p), y transmiten secuencias de vídeo codificadas con H.264. Además, la transmisión se realiza en modo unidifusión (*unicast*), no en multidifusión (*multicast*), más propio de redes vehiculares. De ahí que sea de interés analizar el comportamiento de la transmisión de vídeo en escenarios concretos de redes vehiculares utilizando flujos de vídeo codificados con la última generación de compresores de vídeo.

IV. Entorno de pruebas

Con objeto de evaluar la transmisión de vídeo se ha diseñado un entorno vehicular y una serie de secuencias de simulaciones. Esta sección describe el escenario urbano utilizado, la secuencia de vídeo transmitida por un servidor de vídeo o Road-Side Unit (RSU), y la configuración de los distintos experimentos. Todo ello se ha realizado con la ayuda de un entorno de trabajo denominado Video Delivery Simulation Framework over Vehicular Networks (VDSF-VN) [17], que facilita la creación de los archivos necesarios para realizar simulaciones utilizando el simulador OMNeT++ v4.6 [18], junto con el framework Veins (VEhicles In Network Simulation) v4.4 [19], y el simulador de tráfico SUMO (Simulation of Urban MObility) v0.25.0 [20]. Por otro lado, también permite realizar las tareas previas de codificación de secuencias de vídeo y generación de trazas de vídeo que se utilizarán durante la simulación, así como la posterior evaluación de las métricas de la calidad percibida por un receptor tras las simulaciones.

A. Escenario

El escenario consiste en un área rectangular de la ciudad de Kiev (Ucrania), de tamaño 2000×2000 m^2.

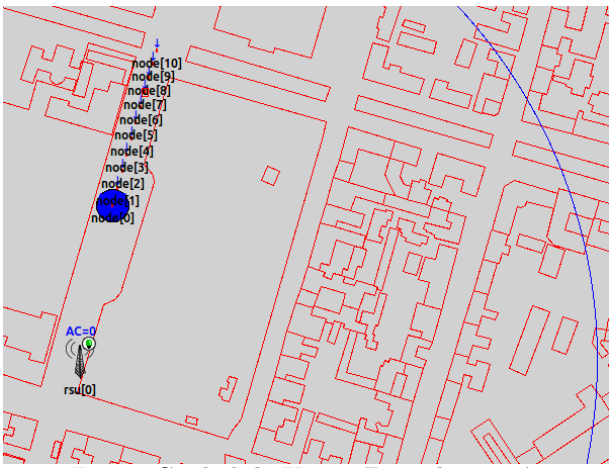


Fig. 3: Ciudad de Kiev - Zona de interés

Se ha colocado una única antena fija o RSU (`rsu[0]`) en medio de una avenida, que transmite una secuencia de vídeo de forma cíclica. El rango de comunicación se ha establecido aproximadamente en 500 m, que es el valor por defecto en Veins, mostrado parcialmente como círculo de color azul en la Figura 3. Los principales parámetros de la simulación se resumen en la Tabla III, y los parámetros de las tarjetas de red se muestran en la Tabla IV.

Durante la simulación, un vehículo cliente de vídeo (`node[0]`) viaja a lo largo de la citada avenida, recibiendo la secuencia de vídeo. Siguiendo a éste, otros 10 vehículos (`node[1..10]`) transmiten datos que se consideran tráfico de fondo. Cada uno de ellos transmite de forma constante una cantidad de paquetes por segundo (pps) de 512 bytes, que se incrementa de un experimento a otro para comprobar el efecto de distintos niveles de tráfico: {0, 12, 25, 50, 75} pps, sumando un total de {0, 120, 250, 500, 750} pps entre todos los vehículos, equivalente a {0, 491, 1024, 2048, 3072} Kbps.

La simulación dura 340s, tiempo suficiente para que los vehículos, que se mueven a una velocidad máxima de 14 m/s (50 km/h), recorran toda la avenida. Sin embargo, nuestra zona de interés está en las proximidades de la antena, en concreto consideramos el intervalo comprendido entre t=[180..190]s, tiempo necesario para la recepción de una secuencia de vídeo completa (10s), que se describe a continuación.

B. Secuencia de vídeo

La secuencia de vídeo transmitida por el RSU es 'BasketballDrill', que pertenece al HEVC Common Test Conditions Set [21]. Como se muestra en la Tabla V, tiene una resolución de 832 × 480 píxeles, una longitud de 250 frames, y una tasa de 25 frames por segundo (fps), representando 10 segundos de vídeo. La secuencia original tiene 500 frames de longitud a una tasa de 50 fps, pero ha sido submuestreada a 25 fps con objeto de reducir el ancho de banda requerido. La secuencia fue codificada con el estándar HEVC, utilizando el software de referencia HEVC Test Model (HM) [21].

Se han utilizado tres modos de codificación para los experimentos (Tabla VI), variando la estructura de cada grupo de imágenes, o Group of Pictures (GoP). En el modo All Intra (AI), todos los frames de la secuencia de vídeo se codifican como frames de tipo I, esto es, ningún frame referencia a otro. Como todos los frames son codificados de forma independiente, el modo AI es el modo de codificación de vídeo más robusto. En el otro extremo está el modo Low-delay P (LP), en el que el primer frame se codifica como un frame de tipo I, y el resto como frames de tipo P, utilizando cuatro frames previamente codificados como referencia. Este modo es muy eficiente en cuanto a compresión debido al uso de estimación de movimiento y compensación, pero es muy sensible a pérdida de paquetes debido a la interdependencia entre frames. Por ello, se utilizan otros modos en los que se inserta periódicamente frames de tipo I (*intra-refresh*) con objeto de evitar la propagación de error temporal. Además del modo AI, en este trabajo se han utilizado los modos de codificación IP7 y LPI4. El modo IP7 es similar al modo LP pero cada frame de tipo I va seguido de siete frames de tipo P. Por último, el modo LPI4 también es una variación del modo LP en el que se inserta un frame de tipo I cada cuatro frames.

Mediante el valor Quantization Parameter (QP) se puede controlar el nivel de calidad o el ancho de banda (*bitrate*) necesario para la transmisión del vídeo codificado. Cuanto mayor valor de QP, mayor compresión se conseguirá, requiriendo un menor espacio de almacenamiento y menor *bitrate* para su transmisión, a costa de una calidad menor. Para cada uno de los modos se estableció el valor de QP para el que se consiguiera, de forma aproximada, la misma calidad de la secuencia de vídeo codificada (PSNR≈36

TABLA III: Parámetros de simulación.

Parámetro	**Valor**
Area de simulación	2000 × 2000 m^2
Tiempo de simulación	340s
N. RSUs	1
N. vehículos cliente	1
N. vehículos tráf. fondo	10
Tasa tráf. fondo	{0, 12, 25, 50, 75} pps
Velocidad máxima	14 m/s (50 km/h)

TABLA IV: Parámetros PHY/MAC.

Parámetro	**Valor**
Frecuencia portadora	5.890 GHz
Modelo de propagación	SimpleObstacleShadowing
Bitrate	18 Mbps
Potencia TX	20 mW
Sensibilidad RX	−89 dBm
Rango comunicación	510.87 m
Tamaño colas MAC	0 (infinito)

TABLA V: Secuencia de vídeo.

Parámetro	Valor
Nombre	BasketballDrill
Resolución	832 × 480 pixels
Duración	10 s
Longitud	250 frames
Tasa	25 fps

TABLA VI: Modos de codificación.

Modo	Estructura	Descripción
AI	IIIIIIII ...	Cada imagen es un I-frame (All Intra)
I7P	IPPPPPPP ...	I-frame seguido de siete P-frames
LPI4	IPPP IPPP ...	Similar a LP pero insertando un I-frame cada cuatro frames

TABLA VII: Valores QP para obtener PSNR≈36 dB.

Modo	QP	Tiles	Bitrate (Kbps)	PSNR (dB)
AI	31	1	3437	35.863
AI	31	16	3656	35.862
I7P	29	1	1467	36.071
I7P	29	16	1604	36.064
LPI4	29	1	1626	36.045
LPI4	29	16	1786	36.034

dB). La Tabla VII muestra el *bitrate* requerido para los casos extremos de cada modo (1 y 16 tiles por frame). Como se puede observar, para una misma calidad percibida, el modo AI es el que requiere un mayor ancho de banda para su transmisión, seguido del LPI4, y, por último, el modo I7P. Esto es debido a que los frames de tipo I son los que requieren almacenar más información al no depender de otros frames, por lo que cuanto mayor proporción de frames de tipo I, mayor ancho de banda será necesario.

Puesto que un frame puede ser mayor que el tamaño máximo de paquete de datos que la red puede transmitir o Maximum Transmission Unit (MTU), para su transmisión a través de la red es necesario encapsular cada frame en una serie de paquetes del protocolo Real-time Transport Protocol (RTP) [22] [23]; a este proceso se le denomina *paquetización* (Fig. 4). En los experimentos realizados, todos los paquetes de un mismo frame tienen la misma prioridad; por ejemplo, si un frame es elegido para tener una prioridad AC(2), todos los paquetes en los que se dividirá para su transmisión tendrán asignada esa prioridad.

En un trabajo previo [24] se utilizaron 9 modos de codificación distintos, estudiando el efecto de dividir los frames de vídeo en fragmentos (*tiles*). En dichos experimentos no se utilizó calidad de servicio (QoS), es decir, en todos los casos se asignó la prioridad AC(0) a todos los paquetes, ya sea de vídeo o de tráfico de fondo. En este trabajo se han seleccionado los tres modos mencionados (AI, I7P, LPI4), con objeto de comprobar el resultado de aplicar calidad de servicio (QoS), asignando la prioridad AC(2) a ciertos paquetes de vídeo, y estudiar cómo afecta todo ello al resto de tráfico no prioritario (tráfico de fondo).

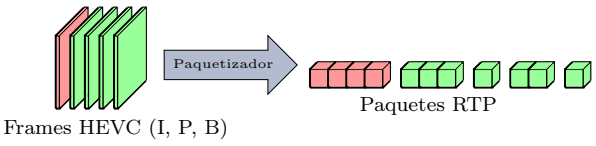


Fig. 4: Paquetización de frames

En los experimentos se han considerado una gran cantidad de combinaciones de parámetros, dependiendo de los modos de codificación y valores QP seleccionados, el número de *tiles* por frame, y el nivel de tráfico de fondo. Todo ello se resume en los siguientes puntos:

- Modos de codificación (×3): {AI, I7P, LPI4} (ver Tabla VI).
- Valores QP (×1): para cada modo se utiliza el valor de QP fijo que permite obtener la calidad de vídeo deseada (PSNR ≈ 36 dB) cuando se usa 1 tile por frame (ver Tabla VII).
- Número de tiles por frame (×7): siete valores se han considerado para codificar cada *bitstream*: {1, 2, 4, 6, 8, 10, 16}. El tamaño de estas particiones se puede especificar con el número de filas y columnas (uniforme), o con el número de Coding Tree Units (CTUs) por cada fila y columna (no-uniforme). En este trabajo se han considerado los siguientes patrones uniformes de tiles: {1 × 1, 1 × 2, 2 × 2, 2 × 3, 2 × 4, 2 × 5, 4 × 4}, respectivamente.
- Tasa de tráfico de fondo (×5): {0, 12, 25, 50, 75} pps de 512 bytes.

Por tanto, todas estas combinaciones hacen un total de 21 *bitstreams* diferentes (3×1×7), con los diferentes niveles de tráfico de fondo (×5), haciendo un total de 105 simulaciones por cada experimento. Se han realizado un total de 10 experimentos, que pueden clasificarse en dos grupos:

1. Grupo 1: asignando una mayor prioridad AC(2) a un porcentaje de los frames de tipo I únicamente (×5): P={0, 25, 50, 75, 100} %; al resto (frames P y B) se le asigna la misma prioridad que el tráfico de fondo, AC(0).
2. Grupo 2: asignando una mayor prioridad AC(2) a un porcentaje de todos los paquetes de vídeo, independientemente del tipo de frame al que corresponda (×5): P={0, 25, 50, 75, 100} %.

El primer grupo de experimentos está motivado en el hecho de que la pérdida de un frame de tipo I es más grave que la pérdida de otros tipos de frame, debido a las interdependencias existentes entre frames. En este caso, puesto que hay paquetes de diferentes

prioridades, tendremos paquetes que al llegar a la capa MAC se clasificarán en distintas colas, y los de menor prioridad tendrán que esperar más que otros paquetes de frames posteriores en el tiempo de reproducción, pero que tengan mayor prioridad. Por ello, es posible que el receptor reciba los paquetes desordenados, por lo que se almacenarán en una memoria intermedia (*buffer*) antes de su reproducción. Sin embargo, también es posible que los paquetes de los frames de baja prioridad puedan sufrir de latencias importantes, por lo que en aplicaciones en tiempo real o sensibles al retardo se establece un retardo máximo para descartar aquellos paquetes que hayan sido recibidos con un retardo mayor.

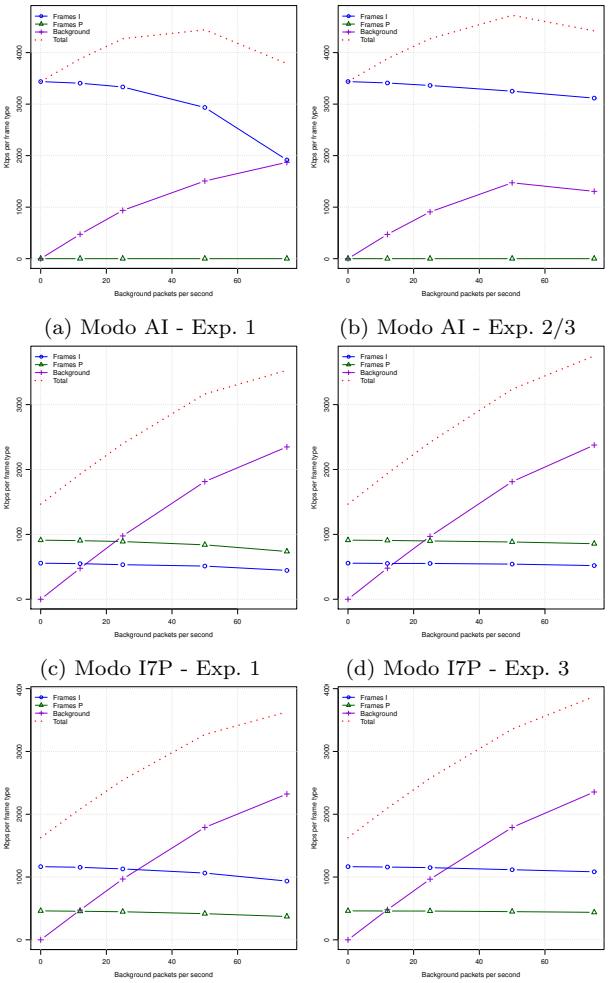
En la Sección V se muestran exclusivamente los resultados de los casos P={0, 100} % por cuestiones de espacio, y a que las otras probabilidades son casos intermedios. Por tanto, y con objeto de facilitar la explicación de resultados, destacamos los resultados de 3 experimentos:

- Experimento 1: asignando la prioridad AC(0), tanto a los paquetes de vídeo como a los de tráfico de fondo (sin QoS).
- Experimento 2: asignando la prioridad AC(2) a todos los frames de tipo I únicamente (P=100 %), y asignando al resto (frames P y B, y tráfico de fondo) la prioridad AC(0).
- Experimento 3: asignando la prioridad AC(2) a todos los paquetes de vídeo (P=100 %), independientemente del tipo de frame al que correspondan. Este caso sería equivalente al mecanismo EDCA del IEEE 802.11e, que especifica que a todos los paquetes de vídeo se les debe asignar la prioridad AC(2).

V. Resultados

Para comparar el rendimiento de los tres experimentos definidos se han ejecutado las correspondientes simulaciones, en las que se ha recogido, entre otras, estadísticas a nivel aplicación (APP), como el número de paquetes transmitidos (Load), los paquetes recibidos (Throughput o Goodput), los paquetes perdidos, el ratio de paquetes enviados respecto a los recibidos o Packet Delivery Ratio (PDR), el retardo o End-to-End Delay (EED), la variación del retardo o *jitter*; también a nivel MAC y físico (PHY), como la ocupación de las colas de cada Access Category (AC), el ratio de ocupación del canal, o el número de colisiones en el medio. Por último, también se calculan métricas objetivas para evaluar la calidad de las secuencias de vídeo recibidas, como el Peak Signal-to-Noise Ratio (PSNR).

Comenzamos mostrando el Throughput a nivel aplicación (Goodput) alcanzado con los tres modos para los tres experimentos. En general, cuando no se utiliza QoS (Exp. 1), el tráfico de vídeo es afectado en gran medida al aumentar el tráfico de fondo, siendo especialmente significativo para el modo AI (Fig. 5a), que, para el mayor nivel de tráfico de fondo, baja un 44.28 % (unos 1.5 Mbps de pérdida). En cuanto al tráfico de fondo, también baja un porcentaje similar



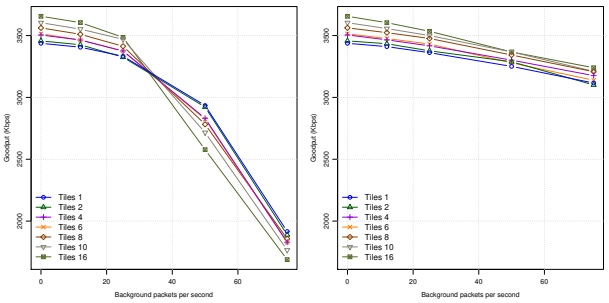
(a) Modo AI - Exp. 1 (b) Modo AI - Exp. 2/3

(c) Modo I7P - Exp. 1 (d) Modo I7P - Exp. 3

(e) Modo LPI4 - Exp. 1 (f) Modo LPI4 - Exp. 3

Fig. 5: Throughput para 1 tile y tráfico de fondo

(39.13 %), ya que ambos tipos de tráfico, al tener la misma prioridad, compiten en el acceso al medio en igualdad de condiciones. En los Exp. 2 y 3, que para el modo AI son idénticos puesto que sólo hay frames de tipo I (Fig. 5b), puede apreciarse que el tráfico de vídeo se ve mucho menos afectado por tener una mayor facilidad a la hora de acceder al canal, teniendo sólo un 9.34 % de pérdida (casi 35 puntos menos), en detrimento del tráfico de fondo, que en el peor caso baja un 57.49 % (13 puntos más que cuando no se utiliza QoS). Vemos también que según se aumenta el tráfico de fondo se va haciendo un mayor uso del canal (sumando el Throughput obtenido tanto por



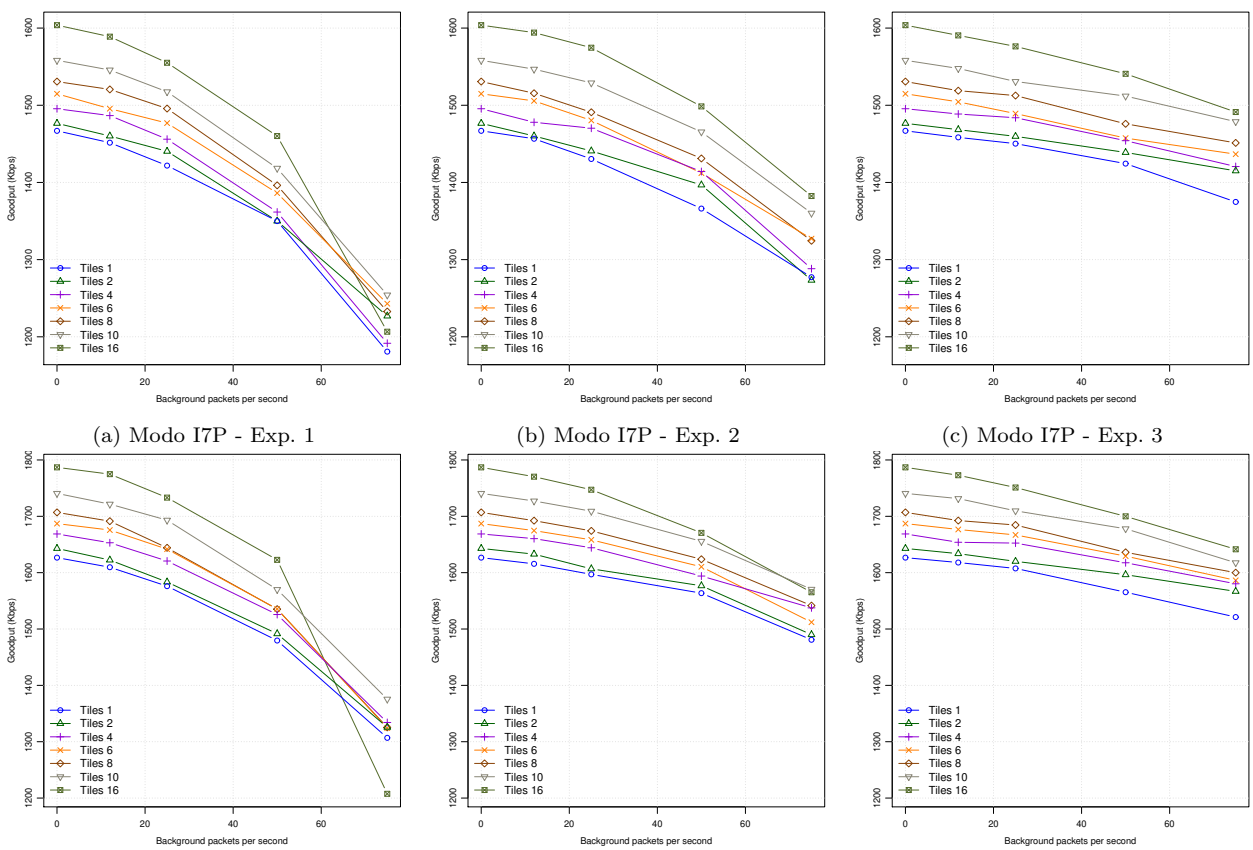
(a) Modo AI - Exp. 1 (b) Modo AI - Exp. 2/3

Fig. 6: Throughput (todos los tiles) - Modo AI

el flujo de vídeo como por el tráfico de fondo de todos los vehículos), llegando al punto máximo para 50 pps; a partir de ahí el canal se satura, y el intentar transmitir un mayor número de paquetes consigue el efecto contrario por aumentar el número de colisiones y periodos de espera para acceso al canal (*backoff*). También se aprecia que al utilizar QoS aumenta la ocupación máxima del canal en ese punto, pasando de los 4.44 Mbps sin QoS a un máximo de 4.72 Mbps (6 % de incremento). Este incremento es aún mayor para 75 pps, pues se alcanza un total de 3.78 Mbps sin QoS, frente a 4.42 Mbps con QoS (16.83 % de incremento).

En cuanto a los modos I7P y LPI4, las pérdidas son aproximadamente la mitad que con el modo AI debido a la menor tasa necesaria para la transmisión de la secuencia de vídeo codificada en este modo, estando el canal más libre. En la Figura 5, el tráfico de vídeo se muestra desglosado en los dos tipos de frames que contienen, I y P. Para el caso de I7P y de mayor nivel de tráfico de fondo (75 pps), en el Exp. 1 (Fig. 5c) el Throughput correspondiente a ambos tipos de frames cae aproximadamente lo mismo (∼19 %), valor parecido al del tráfico de fondo (23.6 %). En el Exp. 2 (no mostrado) los frames de tipo I, al tener más prioridad, apenas sufren una bajada de Throughput (4.68 %), mientras que los frames de tipo P siguen teniendo pérdidas similares al caso anterior (18 %). En el Exp. 3 (Fig. 5d) ambos tipos de frames se mantienen bastante estables, bajando un 6.83 % y 5.93 %, respectivamente, para el mayor nivel de tráfico de fondo. Los resultados del modo LPI4 son muy similares al I7P en todos los experimentos y valores que se acaban de detallar (Fig. 5e y 5f). En cuanto al tráfico de fondo, las pérdidas para los dos modos I7P y LPI4 en ese punto también se mantienen en valores muy parecidos en los 3 experimentos, en torno al 22 %..24 %. Por tanto, vemos que la asignación de prioridad a los distintos frames de vídeo tiene efectos muy positivos a nivel de estadísticas de red, sin perjudicar demasiado al resto de tráfico de menor prioridad existente en la red.

A continuación analizamos el efecto de fragmentar los frames (*tiles*); cuanto mayor número de tiles por frame se utilicen a la hora de codificar la secuencia de vídeo, mayor *bitrate* será necesario (Tabla VII), debido a que se pierde algo de eficiencia en la codificación. Por ejemplo, para el modo AI, el *bitrate* necesario para 1 tile es 3437 Kbps, mientras que para 16 tiles es de 3656 Kbps. Esto provoca que el canal se sature antes según se aumenta el tráfico de fondo, y en caso de no utilizar prioridad, el tráfico de vídeo se ve bastante afectado debido a las colisiones que se producen. En la Figura 6a se puede observar este efecto para el modo AI, ya que, a partir de un determinado nivel de tráfico de fondo, se invierte totalmente el orden de las curvas correspondientes al Throughput de los distintos casos, y las pérdidas llegan hasta un 44.25 % para 1 tile, y 53.84 % para 16 tiles. Sin embargo, al utilizar QoS las curvas se mantienen ordenadas al reducirse los paquetes perdidos hasta una media de 10.36 % para cualquier número



(a) Modo I7P - Exp. 1 (b) Modo I7P - Exp. 2 (c) Modo I7P - Exp. 3

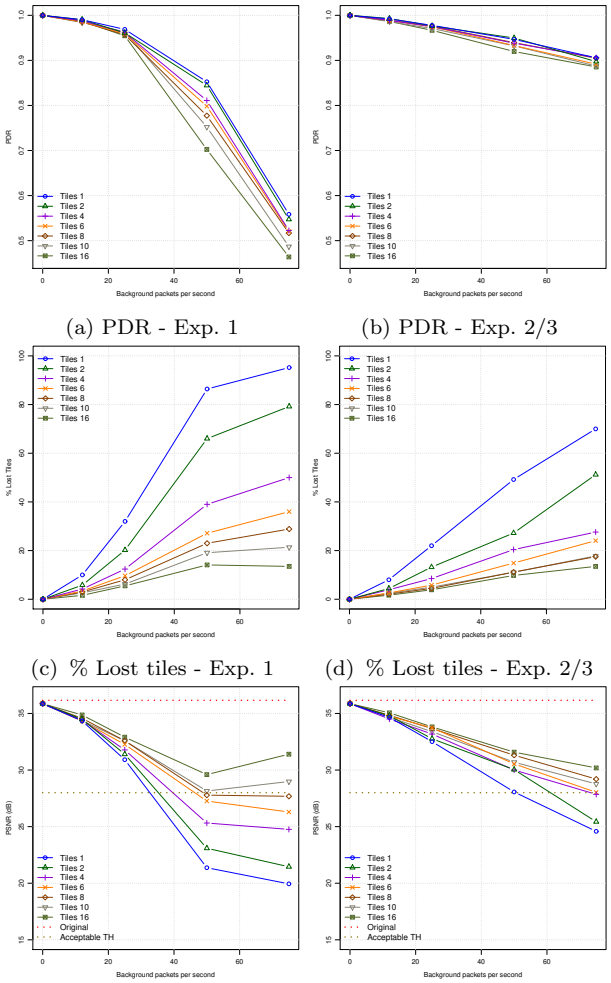
(d) Modo LPI4 - Exp. 1 (e) Modo LPI4 - Exp. 2 (f) Modo LPI4 - Exp. 3

Fig. 7: Throughput (todos los tiles) - Modos I7P y LPI4

de tiles (Fig. 6b).

En cuanto a los modos I7P y LPI4 se observa algo parecido, pero debido a que estos modos requieren un menor *bitrate* que el modo AI, la red sólo se satura para el caso de utilizar 16 tiles por frame. En el modo I7P las pérdidas para el caso de mayor tráfico de fondo se quedan entre 19.56 % para 1 tile y 24.75 % para 16 tiles (Fig. 7a). Igual que para el caso AI, al utilizar QoS (Exp. 2 y 3) las curvas se mantienen ordenadas y tienen similar tendencia, es decir, que el aumento del tráfico de fondo les afecta por igual. Para el modo I7P, en el Exp. 2 las pérdidas están en torno al 13.4 % para cualquier número de tiles por frame utilizado (Fig. 7b), mientras que en el Exp. 3, se quedan en torno al 6.5 % (Fig. 7c). Por último, en el modo LPI4 las pérdidas sin QoS para 1 tile son muy similares al modo I7P (19.67 %), pero para 16 tiles sube hasta 32.46 % debido a la mayor proporción de frames de tipo I y, por tanto, su mayor *bitrate* (Fig. 7d). Al utilizar las dos modalidades de QoS, nuevamente las curvas se mantienen ordenadas, y las pérdidas bajan bastante. En este caso, las pérdidas se quedan en torno al 10.7 % para el Exp. 2 (Fig. 7e) y entorno al 7.3 % para el Exp. 3 (Fig. 7f), siempre algo menores para 1 tile, y algo mayores para 16 tiles.

Para analizar el porcentaje de paquetes entregados correctamente se utiliza el valor del Packet Delivery Ratio (PDR). Las Figuras 8a y 8b muestran el PDR para el modo AI; a partir de ellas se observa cómo baja el número de paquetes entregados, tanto al aumentar el tráfico de fondo, como al aumentar el número de tiles por frame, debido al incremento en el ancho de banda necesario para su transmisión. Sin embargo, al incrementar el número de tiles por frame, a pesar de tener un mayor número de pérdidas, el porcentaje de tiles perdidos es menor, lo que implica que la calidad del vídeo reconstruido sea mayor. Por ejemplo, en el modo AI, las pérdidas de paquetes son especialmente notables en el Exp. 1 (sin QoS), donde las curvas del PDR muestran una fuerte bajada cuando aumenta el tráfico de fondo; para el caso de 1 tile por frame se produce una pérdida de paquetes de un 44 %, y para el caso de 16 tiles, un 53 % (Fig. 8a). Estas pérdidas tienen como consecuencia que no sea posible decodificar un 95.2 % de los tiles para 1 tile por frame, mientras que en el caso de utilizar 16 tiles por frame, tan sólo un 13.5 % (Fig. 8c). Esto es debido a que la pérdida de un único paquete en un tile hace que el tile completo no pueda decodificarse, que en el caso de 1 tile por frame significa el frame completo. El resultado en ese caso es un vídeo recibido de una calidad muy pobre, con un valor PSNR de menos de 20 dB, mientras que si se utilizan 10 ó 16 tiles, la calidad supera el umbral considerado como aceptable (28 dB) en todos los casos; en concreto, para 16 tiles supera los 31 dB (Fig. 8e). Siguiendo con el modo AI, en el Exp. 2 (o 3) apenas hay diferencia en los valores del PDR para cualquier número de tiles por frame. Sin embargo, el número de tiles perdidos y la calidad final del vídeo reconstruido sí que muestran una gran diferencia en ambos casos. Con 1 tile se pierde



(a) PDR - Exp. 1 (b) PDR - Exp. 2/3

(c) % Lost tiles - Exp. 1 (d) % Lost tiles - Exp. 2/3

(e) PSNR - Exp. 1 (f) PSNR - Exp. 2/3

Fig. 8: Modo AI - PDR (arriba), % de paquetes perdidos (centro), y PSNR (abajo)

el 9.44 % de los paquetes (Fig. 8b), que se traduce en una pérdida del 70 % de los tiles (Fig. 8d), y se obtiene un PSNR de 24.58 dB (Fig. 8f). Sin embargo, con 16 tiles por frame, a pesar de que se pierde un porcentaje ligeramente mayor de paquetes (11.46 %), eso conlleva la pérdida de sólo el 13.5 % de los tiles, obteniendo una calidad final de 30.19 dB.

La Tabla VIII muestra el valor PSNR obtenido con el modo AI para todas las combinaciones de tráfico de fondo (BGT pps) y número de tiles por frame (de 1 a 16), para el Exp. 1 (arriba) y 2/3 (abajo), corres-

TABLA VIII: Modo AI - PSNR Exp. 1 y 2/3 (dB).

BGT	1	2	4	6	8	10	16
0	35,86	35,87	35,86	35,87	35,86	35,86	35,86
12	34,34	34,49	34,41	34,33	34,57	34,36	34,87
25	30,91	31,40	31,78	32,26	32,59	32,59	32,91
50	21,37	23,09	25,32	27,26	27,78	28,15	29,60
75	19,95	21,46	24,76	26,29	27,68	28,99	31,40

BGT	1	2	4	6	8	10	16
0	35,86	35,87	35,86	35,87	35,86	35,86	35,86
12	34,65	34,80	34,51	34,75	34,84	34,66	35,05
25	32,51	32,76	33,18	33,73	33,66	33,39	33,81
50	28,06	30,04	30,00	30,54	31,31	30,69	31,58
75	24,58	25,43	27,85	28,03	29,20	28,77	30,19

TABLA IX: Modo I7P - PSNR Exp. 1, 2 y 3 (dB).

BGT	1	2	4	6	8	10	16
0	36,07	36,08	36,08	36,08	36,07	36,07	36,06
12	30,53	32,57	33,76	33,39	34,31	33,63	33,71
25	25,62	28,10	29,69	31,81	31,55	30,54	30,42
50	20,77	21,01	23,44	24,22	24,28	25,25	25,19
75	16,77	17,61	19,23	20,33	21,38	20,26	21,63

BGT	1	2	4	6	8	10	16
0	36,07	36,08	36,08	36,08	36,07	36,07	36,06
12	33,41	32,21	33,39	34,50	34,01	34,12	34,29
25	29,46	29,09	30,49	30,96	31,11	31,98	31,35
50	23,90	25,11	25,96	26,51	27,48	26,72	26,15
75	19,34	20,92	22,38	22,85	24,20	23,45	24,04

BGT	1	2	4	6	8	10	16
0	36,07	36,08	36,08	36,08	36,07	36,07	36,06
12	33,85	34,20	35,00	34,22	34,38	34,52	34,60
25	32,41	34,02	33,56	31,43	32,57	32,68	32,35
50	26,48	29,34	30,40	28,82	29,60	30,53	29,14
75	22,83	26,92	26,91	26,43	26,86	28,04	26,28

TABLA X: Modo LPI4 - PSNR Exp. 1, 2 y 3 (dB).

BGT	1	2	4	6	8	10	16
0	36,05	36,05	36,05	36,05	36,05	36,05	36,03
12	32,90	33,22	34,10	34,52	34,51	34,09	34,77
25	27,32	28,65	31,06	31,71	30,78	32,99	32,58
50	20,89	23,41	25,89	25,66	26,56	26,45	27,29
75	17,14	19,65	20,57	21,69	21,89	22,61	23,90

BGT	1	2	4	6	8	10	16
0	36,05	36,05	36,05	36,05	36,05	36,05	36,03
12	34,12	34,66	34,94	35,08	34,93	34,71	34,58
25	31,34	31,12	33,15	32,52	32,84	33,22	32,35
50	28,59	28,48	28,94	30,05	29,05	29,34	28,81
75	21,36	24,79	26,46	25,35	26,08	26,43	25,55

BGT	1	2	4	6	8	10	16
0	36,05	36,05	36,05	36,05	36,05	36,05	36,03
12	34,06	35,24	33,50	34,67	34,57	35,11	34,45
25	32,03	32,87	33,84	33,92	33,50	33,56	32,96
50	27,40	29,82	30,96	30,54	30,56	30,95	29,58
75	22,28	27,50	28,43	28,38	28,63	27,62	28,17

pondientes a las Figuras 8e y 8f, respectivamente. Si el valor de PSNR es mayor de 32 dB (valor muy bueno) se marca en color verde, si es mayor de 28 dB (valor aceptable) se marca en amarillo, si está por debajo de ese valor y hasta 24 dB (valor deficiente) en naranja, y para valores inferiores (muy deficientes) en rojo.

En cuanto al modo IP7, al ser el de menor refresco Intra, no es muy robusto. A partir de un nivel de tráfico de fondo moderado los resultados ya no son aceptables (Tabla IX). Por último, el modo LPI4 es más robusto que el I7P, obteniendo unos resultados bastante aceptables (Tabla X), muy parecidos a los del modo AI (Tabla VIII).

Para terminar, analizamos la calidad del vídeo de los tres modos en cada uno de los tres experimentos para el caso de mayor tráfico de fondo (75 pps) variando el número de tiles por frame (Fig. 9). Se puede deducir que al incrementar el número de tiles hasta un número de 6 u 8 la calidad experimenta una mejora significativa en todos los modos, compensando el exceso de ancho de banda requerido para codificar la secuencia de vídeo. Sin embargo, para valores mayores las mejoras son mínimas. Los mejores resultados se obtienen en el experimento 3, en el que los modos AI y LPI4, y para un número de tiles igual o superior a 6 tiles alcanzan una calidad aceptable; en cuanto al modo I7P, aunque el uso de tiles también proporciona una mejora, no es suficiente cuando hay niveles moderados o altos de tráfico de fondo.

VI. Conclusión y Trabajos futuros

Se han realizado varios experimentos combinando diversas técnicas con objeto de mejorar la calidad del vídeo transmitido en redes vehiculares, como son el refresco de frames de tipo I (*intra-refresh*), el número de tiles por frame, y la calidad de servicio. Se han utilizado varios modos de codificación variando la frecuencia de refresco de frames de tipo I, llegando a la conclusión de que es fundamental el uso de frames de tipo I frecuentemente, como en los modos AI y LPI4, descartando el uso del modo I7P. Usar un mayor número de tiles por frame incrementa la calidad del vídeo reconstruido, siendo 6 u 8 un número que mejora notablemente la calidad, sin incrementar demasiado el ancho de banda necesario para su transmisión. Además, todo ello se ha combinado con el uso de la calidad de servicio (QoS) mediante la asignación de más prioridad a ciertos frames de la secuencia de vídeo. Sin QoS, solamente el modo AI supera el umbral de calidad aceptable a partir de 10 tiles por frame; los modos I7P y LPI4 no lo alcanzan en ningún caso. Mediante el uso de QoS, asignando la prioridad AC(2) a los frames de tipo I, los resultados indican que aunque la calidad mejora, no es suficiente para llegar al umbral aceptable en esos modos. En cambio, asignando la prioridad AC(2) a todos los frames de vídeo, el modo LPI4 también llega a dicho umbral, no siendo así para el modo I7P, que, aunque mejora su calidad, sigue sin ser suficiente. Como conclusión final, el uso de los modos AI o LPI4, con 6 u 8 tiles por frame, y asignando una prioridad AC(2) a todos los frames de vídeo es la alternativa que proporciona los mejores resultados.

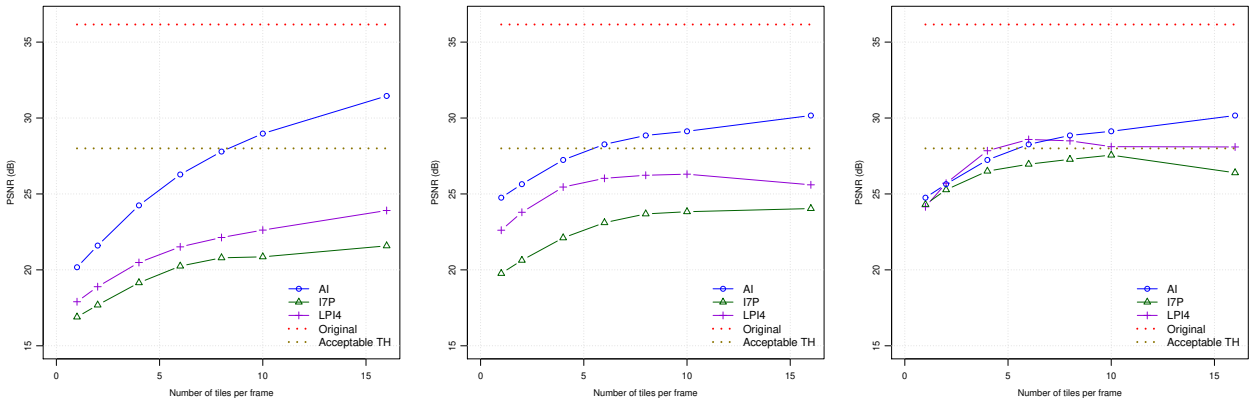
Como continuación de este trabajo se propone el uso técnicas adaptativas, que tengan en cuenta el nivel de saturación de la red, y el tamaño de las distintas colas del MAC, así como el uso de otras técnicas de protección de errores.

Agradecimientos

Este trabajo ha sido financiado por el Ministerio de Ciencia, Innovación y Universidades (Ref. RTI2018-098156-B-C54) cofinanciado con fondos FEDER (MINECO/FEDER/UE), así como por la Convocatoria de Ayudas y Bolsas de viaje para la difusión de resultados de investigación en el marco del programa de doctorado TECNIT (RR 2426/18 UMH).

Referencias

[1] Ali Tufail, Syed Ali Khayam, Muhammad Taqi Raza, Amna Ali, and Ki-Hyung Kim, “An Enhanced Backbone-Assisted Reliable Framework for Wireless Sensor Net-



(a) Exp. 1 (b) Exp. 2 (c) Exp. 3

Fig. 9: PSNR Todos los modos - Variando número de tiles por frame

works," *Sensors (Basel, Switzerland)*, vol. 10, pp. 1619–51, 03 2010.

[2] Joint Collaborative Team on Video Coding (JCT-VC), "High Efficiency Video Coding (HEVC)," ITU-T Recommendation H.265, 2013.

[3] "Advanced Video Coding (AVC) for generic audiovisual services," ITU-T Recommendation H.264, 2003.

[4] D. B. Rawat and D. C. Popescu and, "Performance enhancement of EDCA access mechanism of IEEE 802.11e wireless LAN," in *2008 IEEE Radio and Wireless Symposium*, Jan 2008, pp. 507–510.

[5] Hua Zhu, Ming Li, , Imrich Chlamtac, and B. Prabhakaran, "A survey of quality of service in IEEE 802.11 networks," *IEEE Wireless Communications*, vol. 11, no. 4, pp. 6–14, Aug 2004.

[6] "IEEE Standard for Information technology–Telecommunications and information exchange between systems Local and metropolitan area networks–Specific requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications," *IEEE Std 802.11-2016*, pp. 1–3534, Dec 2016.

[7] "IEEE Standards for Information Technology – IEEE Std 802.11e – Draft Supplement to Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications: Medium Access Control (MAC) Quality of Service Enhancements," online, 2005.

[8] "IEEE Standard for Wireless Access in Vehicular Environments (WAVE) – Multi-Channel Operation," *IEEE Std 1609.4-2016 (Revision of IEEE Std 1609.4-2010)*, pp. 1–94, March 2016.

[9] "IEEE Standard for Information technology – Local and metropolitan area networks – Specific requirements – Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications Amendment 6: Wireless Access in Vehicular Environments," *IEEE Std 802.11p-2010*, pp. 1–51, July 2010.

[10] A. Ksentini, M. Naimi, and A. Gueroui, "Toward an improvement of H.264 video transmission over IEEE 802.11e through a cross-layer architecture," *IEEE Communications Magazine*, vol. 44, no. 1, pp. 107–114, Jan 2006.

[11] R. MacKenzie, D. Hands, and T. O'Farrell, "QoS of Video Delivered over 802.11e WLANs," in *2009 IEEE International Conference on Communications*, June 2009, pp. 1–5.

[12] I. Ali, M. Fleury, S. Moiron, and M. Ghanbari, "Enhanced prioritization for video streaming over wireless home networks with IEEE 802.11e," in *2011 IEEE International Symposium on Broadband Multimedia Systems and Broadcasting (BMSB)*, June 2011, pp. 1–6.

[13] Zheng Wan, N. Xiong, N. Ghani, Min Peng, A. V. Vasilakos, and Liang Zhou, "Adaptive scheduling for wireless video transmission in high-speed networks," in *2011 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, April 2011, pp. 180–185.

[14] Naveen Chilamkurti, Sherali Zeadally, Robin Soni, and Giovanni Giambene, "Wireless multimedia delivery over 802.11e with cross-layer optimization techniques," *Multimedia Tools and Applications*, vol. 47, no. 1, pp. 189–205, Mar 2010.

[15] C.-H. Lin, C.-K. Shieh, C.-H. Ke, N. K. Chilamkurti, and S. Zeadally, "An adaptive cross-layer mapping algorithm for MPEG-4 video transmission over IEEE 802.11e WLAN," *Telecommunication Systems*, vol. 42, no. 3, pp. 223, Jul 2009.

[16] Xin-Wei Yao, Wan-Liang Wang, Shuang-Hua Yang, Yue-Feng Cen, Xiao-Min Yao, and Tie-Qiang Pan, "IPB-frame Adaptive Mapping Mechanism for Video Transmission over IEEE 802.11e WLANs," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 2, pp. 5–12, Apr. 2014.

[17] P. Pablo Garrido Abenza, Pablo Piñol Peral, Manuel P. Malumbres, and O. López-Granado, "Simulation Framework for Evaluating Video Delivery Services over Vehicular Networks," in *2018 IEEE 88th Vehicular Technology Conference (VTC-Fall)*, August 2018, pp. 1–5.

[18] András Varga and Rudolf Hornig, "An Overview of the OMNeT++ Simulation Environment," in *Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops*, ICST, Brussels, Belgium, Belgium, 2008, Simutools '08, pp. 60:1–60:10, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[19] Christoph Sommer, Reinhard German, and Falko Dressler, "Bidirectionally Coupled Network and Road Traffic Simulation for Improved IVC Analysis," *IEEE Transactions on Mobile Computing*, vol. 10, no. 1, pp. 3–15, January 2011.

[20] Daniel Krajzewicz, Jakob Erdmann, Michael Behrisch, and Laura Bieker, "Recent Development and Applications of SUMO - Simulation of Urban MObility," *International Journal On Advances in Systems and Measurements*, vol. 5, no. 3&4, pp. 128–138, December 2012.

[21] Joint Collaborative Team on Video Coding (JCT-VC), "HEVC reference software HM (HEVC Test Model) and Common Test Conditions," `https://hevc.hhi.fraunhofer.de`, [Accessed May 11, 2018].

[22] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications," RFC 3550, 2003.

[23] Ye Wang, Yago Sanchez, Thomas Schierl, Stephan Wenger, and Miska Hannuksela, "RTP Payload Format for High Efficiency Video Coding," RFC 7798, 2016.

[24] Pedro Pablo Garrido Abenza, Manuel P. Malumbres, Pablo Piñol, and Otoniel López-Granado, "Source Coding Options to Improve HEVC Video Streaming in Vehicular Networks," *Sensors*, vol. 18, no. 9, 2018.

OPASim: un simulador para redes de interconexión de OPA con soporte de QoS

Javier Cano-Cano[1], Francisco J. Alfaro[1], José L. Sánchez[1], Guillermo Fernández[1] y Francisco J. Andújar[2]

Resumen— **En la carrera por alcanzar computadores exascale, han aparecido nuevas tecnologías como Intel© Omni-Path© (OPA). Estas nuevas tecnologías deben ser probadas, evaluadas y estudiadas. Para tal labor, la técnica más extendida es la simulación. A través de ella, es posible evaluar nuevas propuestas basadas en dichas tecnologías de una forma rápida, eficiente y flexible. Hasta donde sabemos, no existe ninguna herramienta de simulación centrada en OPA. Debido a ello, se ha creado un simulador basado en la información pública acerca de OPA. El simulador, bautizado como OPASim, es capaz de simular una gran variedad de topologías, algoritmos de encaminamiento y políticas de calidad de servicio. OPASim ha sido concebido para ser un simulador rápido, eficiente, flexible y modular. En este articulo se explican algunos detalles del modelo de simulación de OPA, detalles de implementación de dicho modelo en el simulador, mecanismos de calidad de servicio presentes en la tecnología, políticas de calidad de servicio y se muestran algunos resultados preliminares.**

Palabras clave— **Redes de interconexión. Omni-Path. Simulación. Calidad de servicio.**

I. Introducción

El aumento de la demanda de los servicios de supercomputación se estima que llegará a un exaflop en los próximos años. El incremento de rendimiento de los supercomputadores se puede alcanzar mediante: a) el aumento del número de núcleos por nodo sin incrementar el rendimiento individual; b) el incremento del número de nodos sin incrementar el número de núcleos por nodo y c) una combinación de las dos opciones anteriores.

La red de interconexión es un elemento clave en estos sistemas, y un mal diseño puede convertirla en un cuello de botella del sistema. En ese caso, la consecuencia sería la degradación del rendimiento de todo el sistema. El rendimiento de la red de interconexión depende de varios factores (topología, encaminamiento, disposición los elementos, etc.), que deben estudiarse detenidamente por los diseñadores del sistema.

La arquitectura Intel© Omni-Path© (OPA) representa una nueva familia de productos orientados a las redes de interconexión de altas prestaciones. OPA está diseñado para la interconexión de los componentes con CPU y memoria para habilitar una latencia reducida, alto ancho de banda y alta escalabilidad para la venidera generación de sistemas exascale.

[1]Departamento de Sistemas Informáticos, Universidad de Castilla-La Mancha, e-mail: {javier.cano, fco.alfaro, jose.sgarcia,guillermo.fernandez}@uclm.es

[2]Departamento de Informática, Universidad de Valladolid. e-mail: fandujarm@infor.uva.es

La llegada de OPA al mercado marca una de las líneas de productos de redes de interconexión más significativa desde la aparición de InfiniBand [1], [2]. No obstante, hasta el momento OPA no se ha estudiado tanto como otras tecnologías de redes de interconexión. Uno de los motivos principales es que OPA es una tecnología bastante nueva. También influye la poca disponibilidad de herramientas que permitan a los investigadores estudiar sistemas basados en OPA.

La simulación es una de las técnicas más habituales para analizar cualquier arquitectura de redes de interconexión de una forma flexible, barata, rápida y reproducible. En lo que respecta a OPA, hasta donde se sabe, no existe ninguna herramienta, simulador o modelo de simulación disponible. Este hecho dificulta las labores de investigación y, en algunos casos, las impide. En este artículo se presenta un modelo de simulación y un simulador de eventos discretos basado en OPA. Este trabajo está basado en toda la información pública acerca de la arquitectura. No obstante, se han tomado algunas suposiciones debido a que muchos detalles internos no están publicados e Intel no desvela.

El resto del artículo se ha organizado de la siguiente forma: En la sección II se explica y detalla el funcionamiento interno de OPASim y el modelo de simulación OPA. La sección III define todos lo mecanismos de calidad de servicio disponibles en la arquitectura. En la sección IV se presenta una propuesta de calidad de servicio implementada en OPA. En la sección V se incluye una batería de simulaciones cuyo objetivo es mostrar el correcto funcionamiento de OPASim. Finamente, la sección VI recoge las conclusiones de este artículo.

II. Detalles del simulador de OPA

Se ha desarrollado un simulador de redes de interconexión de eventos discretos, capaz de simular el movimiento de paquetes desde los nodos de origen a destino usando un modelo de red de interconexión basado en OPA. Se han modelado los elementos principales de OPA como interfaces de red, switches y enlaces. Además, el simulador ofrece la posibilidad de modificar una gran cantidad de parámetros como el tamaño de las colas, la topología, el mecanismo de encaminamiento, el tamaño de paquetes, etc.

El objetivo principal es obtener una herramienta de simulación tan flexible como sea posible, siguiendo el comportamiento de OPA, orientada a evaluar configuraciones de red basadas en esta tecnología. La herramienta es capaz de ejecutar si-

mulaciones usando una amplia variedad de tipos de tráfico sintético como podrían ser: tráfico uniforme, bit-reversal, bit-complement, etc., y trazas de aplicaciones MPI usando el framework VEF [3]. El rendimiento y la escalabilidad de la red se evalúan usando múltiples métricas: productividad, latencia de la red, latencia nodo-a-nodo, etc.

Por otra parte, OPASim implementa los principales mecanismos de calidad de servicio incluidos en OPA. En la sección IV se presenta una posible técnica de provisión de QoS usando los mecanismos propuestos por OPA.

Como trabajo futuro, se pretende evaluar diferentes propuestas para aquellos elementos de OPA cuyo comportamiento no está documentado de forma pública.

A. Modelo de simulación

El primer paso en el desarrollo de la herramienta ha sido definir un modelo basado en la información pública disponible en [4] y [5]. Y, a parte de este modelo, se ha desarrollado OPASim, un simulador de redes dirigido por eventos, flexible, de código abierto, eficiente y rápido. Esta herramienta simula los detalles específicos de los switches OPA con la suficiente granularidad y precisión.

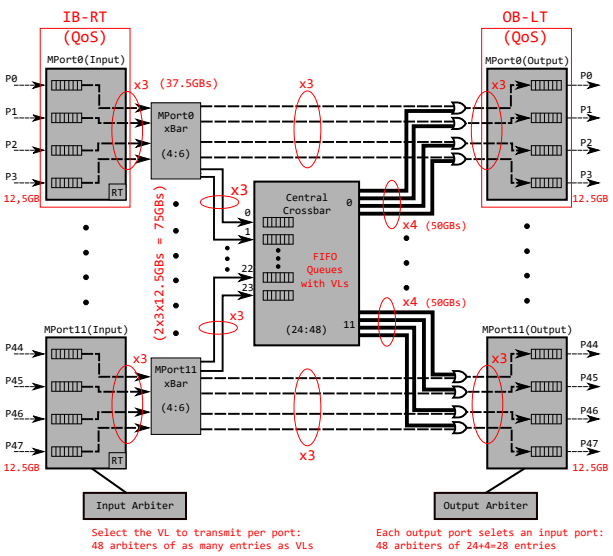


Fig. 1. Diagrama del switch OPA de 48 puertos modelado. Por claridad, los MPorts se han desplegado en los buffers de entrada y salida.

Se asume que cada enlace transmite un flit por ciclo, por tanto el ancho de banda se define en función del número de ciclos de reloj y el tamaño del flit. La figura 1 detalla el modelo de switch de 48 puertos que se ha considerado e implementado en OPASim. Dentro de un switch OPA se puede configurar un rango amplio de anchos de banda. Se define el ancho de banda de los puertos de entrada/salida (12,5 GBps) como referencia. Se entiende por tanto, que un puerto/enlace x3 presenta una aceleración de 3 y puede enviar 3 flits/ciclo. El número de puertos de entrada y salida está representado en la figura 1 como `ENTRADA:SALIDA` en los elementos del conmutador (MPort xBars y Central Crossbar). Por ejemplo, el MPort0 tiene 4 puertos de entrada y 6 de salida (4:6) y el Central Crossbar tiene 24 puertos de entrada y 48 puertos de salida (24:48). Los elementos incluidos en esta figura son los siguientes:

- Buffers de entrada: Almacenan los flits de los puertos de entrada. Hay un buffer de entrada por cada puerto de entrada.
- Unidades de encaminamiento: Determinan el puerto de salida para los flits en los buffers de entrada.
- MPort Xbar: Presenta 4 puertos de entrada, uno por buffer de entrada y 6 de salida: 2 para los buffers del Central Crossbar y 4 para los buffers de salida del propio MPort. Nótese que los enlaces de 75 GBps conectados a los buffers centrales están modelados por 2 enlaces. Cada enlace puede enviar 3 flits/ciclo, lo cual resulta en 2 Links $\times$ 3 $\times$ (12,5 GBps) = 75 GBps.
- Buffers de salida: Almacenan los flits de los puertos de salida. Hay un buffer de salida por cada puerto de salida.
- Árbitros de entrada: Dado un puerto de entrada, selecciona un canal virtual (VL) que participará en la siguiente fase de arbitraje.
- Árbitros de salida: Dado un buffer de salida, éste elige un puerto de entrada que podrá transmitir flits. Un flit puede llegar a un puerto de salida desde un buffer central o desde un buffer de entrada.

Con respecto a los eventos usados en el modelo de simulación, OPASim define los típicos eventos de una arquitectura de switch:

- IB (Input Buffering): Los flits llegan a los puertos de entrada y se almacenan en la cola correspondiente, dependiendo del VL. Si el flit es una cabecera de paquete, se marca como *RT-ready*, indicando que se debe invocar a la unidad de encaminamiento para determinar el puerto de salida. Si el flit no es de cabecera, se marca como *X-ready* y se almacena a la espera de ser transmitido a un buffer de salida o a un buffer central. Un flit tendrá que atravesar el Central Crossbar si el puerto de salida está en otro MPort distinto que el puerto de entrada. Por ejemplo, esto ocurriría en un switch OPA con 48 puertos y 4 puertos por MPort (como el mostrado en la figura 1) si un flit necesita viajar desde el puerto 0 hasta el puerto 5. El puerto de entrada se encuentra conectado al MPort 0, mientras que el puerto de salida está conectado al MPort 1. Así pues, el flit deberá cruzar el Central Crossbar para alcanzar el MPort 1.
- RT (RouTing): Si el flit de cabecera de un paquete está marcado como *RT-ready*, el evento realiza la función de encaminamiento predefinida. El objetivo es determinar a qué puerto del switch se debe enviar el paquete para alcanzar su nodo destino. Después de esto, se marca el flit de cabecera como *VA-SA-ready* y el buffer de entrada que almacena dicho flit será

un candidato para la fase VA-SA. Nótese que este evento solo se aplica a los flits de cabecera, ya que la técnica de conmutación implementada por el switch OPA [4] es *virtual cut-through* [6] y el resto de flits *no-cabecera* siempre siguen al flit de cabecera a través del switch. El usuario puede configurar la función de encaminamiento acorde con la topología seleccionada.

- VA-SA (Virtual Allocator and Switch Allocator): Se trata de un asignador de 2 etapas:
  - Virtual Allocator: Cada árbitro de entrada, con al menos un flit marcado como *VA-SA-ready*, elige un VL que estará autorizado a transmitir un paquete. El árbitro implementado es un árbitro Round-Robin. Nótese que, debido a que los puertos del Central Crossbar tienen VLs, esta fase del asignador también se realiza en los puertos de entrada del Central Crossbar.
  - Switch Allocator: Cada árbitro de salida elige un buffer de salida con un VL asignado en la etapa anterior. Los buffers de entrada seleccionados podrán mover paquetes desde los buffers de entrada a los buffers de salida o buffers centrales, dependiendo de si el puerto destino está en el mismo MPort o no. Los buffers autorizados a transmitir paquetes marcan el flit en las cabezas de los buffers como *X-ready*. Los buffers centrales tienen que arbitrar entre los 4 buffers de entrada que tienen conectados a su MPort. Los buffers de salida tienen que arbitrar entre los 24 buffers centrales y los 4 de su MPort.
- X (Xbar): Una vez que el proceso de asignación se ha completado, los buffers de entrada y los centrales seleccionados en el evento VA-SA, transmiten flits marcados como *X-ready* a los buffers de salida o centrales apropiados. Si un flit se mueve desde un buffer de entrada hasta un buffer central, dicho flit se marca de nuevo como *VA-SA-ready*. El objetivo es que ese flit tenga que participar de nuevo en el proceso de asignación para poder alcanzar el buffer de salida correspondiente. Cuando los flits alcanzan el buffer de salida solicitado, se marcan como *OB-ready*. El ancho de banda depende de la pareja entrada/salida. Desde los MPorts hasta el Central Crossbar, se pueden enviar 3 flits/ciclo y, desde el Central Crossbar hasta los buffers de salida 4 flits/ciclo.
- OB (Output Buffering): Cada buffer de salida con paquetes marcados como *OB-ready*, elige un VL que podrá enviar paquetes al siguiente switch o interfaz de red. El proceso de selección de VLs se puede realizar a través de un árbitro Round-Robin, en caso de no desear calidad de servicio, tablas de arbitraje, algoritmos con garantías de calidad de servicio como DRR [7], ERR [8], etc. Se asigna un VL a cada buffer de salida con paquetes marcados como *OB-ready*, y suficientes créditos como para transmitir al menos un paquete. Cuando un buffer de salida transmite el último flit del paquete (el flit de cola), el buffer libera el VL asignado anteriormente. De esta forma, el buffer tendrá que participar en el proceso de asignación de VLs de nuevo, dando la oportunidad a otros VLs de enviar paquetes. Cada puerto de salida puede enviar 1 flit/ciclo. En este punto, como ya se ha mencionado, se pueden aplicar estrategias de calidad de servicio (QoS) y adelantamiento de paquetes.

En los buffers, el espacio asignado a cada VL se gestiona de forma dinámica. Esto significa que no se reserva un espacio fijo para cada VL. El espacio se divide y consume en función de los requisitos del tráfico. No obstante, se asegura un espacio mínimo y máximo para cada VL. Esta estrategia permite mayor flexibilidad que las asignaciones estáticas y espacios pre-reservados de los buffers [9]. Esta flexibilidad es especialmente útil para tareas de provisión de calidad de servicio.

B. Latencia de las etapas

De acuerdo con la información disponible, un paquete necesita un total del 100 ns para cruzar un switch OPA [4], [5]. Se han analizado todas las posibles fuentes de retraso en el modelo de simulación de OPA y distribuido la latencia total entre todos los pasos que un paquete debe seguir a través de la arquitectura del switch. La figura 2 muestra la distribución de las latencias a través de las diferentes acciones que se realizan.

Se ha considerado la latencia total de un flit de cabecera de paquete desde que se inyecta hasta que se transmite al siguiente switch de esta forma:

- La interfaz de red (NIC) inyecta el flit de cabecera. Esto se encuentra representado por el estado *"NIC injects a flit"* en la figura 2 (arriba a la izquierda).
- El flit suma *INJ* ciclos a la latencia total de su paquete.
- El flit dispara el evento *"Input Buffering and Routing"*. La latencia de almacenar el flit en el buffer de entrada son *SB* ciclos y la latencia del encaminamiento son *RT* ciclos. Hasta este punto, la latencia total acumulada es $INJ + SB + RT$ ciclos.
- Después, el flit puede participar en el proceso de asignación *"Allocation"*. La latencia del asignador son *AT* ciclos. El total ahora es $INJ + SB + RT + AT$ ciclos.
- Los siguientes eventos son *"Cross MPort XBar"* y *"Cross Central XBar"*. En estos eventos, el flit se puede mover directamente a un buffer de salida, o a un buffer central y posteriormente a un buffer de salida. Si el flit se mueve directamente a un buffer de salida, tan solo sumará X ciclos. No obstante, si necesita cruzar el buffer central, sumará $X + AT + X + SB$ ciclos a la latencia final. Esto se debe a que el flit tendrá que

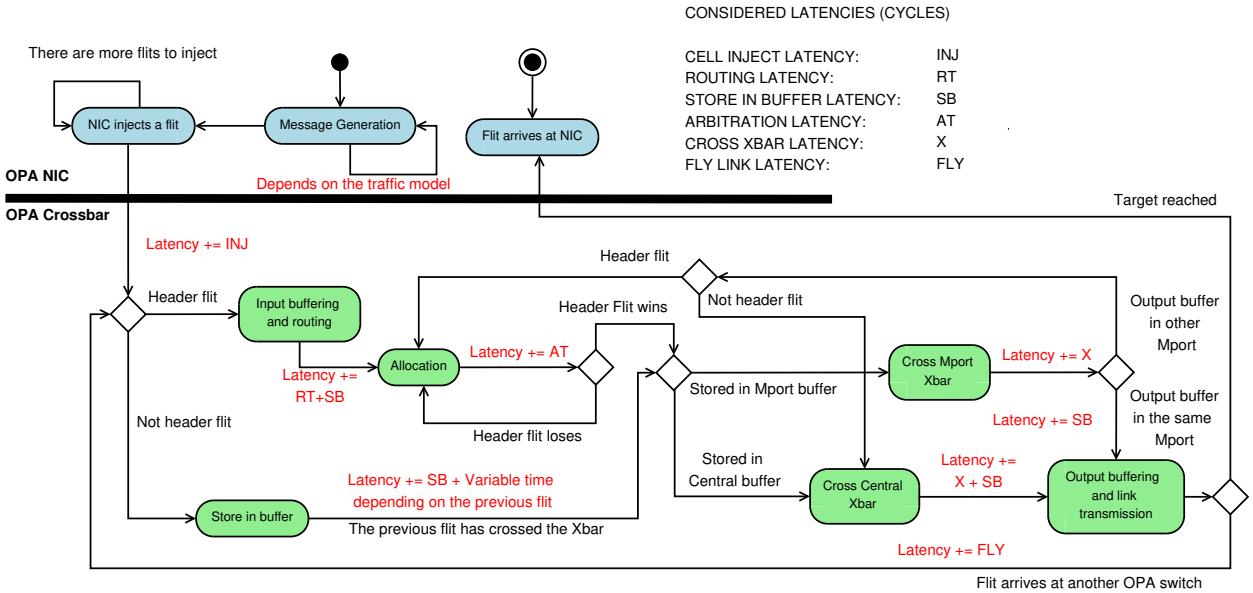


Fig. 2. Distribución de la latencia entre todos los eventos.

cruzar su MPort, participar en un nuevo proceso de asignación (*"Allocation"*), moverse desde el buffer central hasta el buffer de salida y, finalmente, almacenarse en el buffer de salida. En la sección II-A (descripción de los eventos VA-SA y X) se pueden encontrar más detalles acerca de este proceso. Por tanto, la latencia acumulada será $INJ + SB + RT + AT + X + AT + X + SB$ ciclos.

- Nótese que no hay latencia asociada a almacenar flits en los buffer centrales. Se han considerado estos buffers como "perfectos", es decir, no requieren tiempo adicional para almacenar los datos.
- Finalmente, el flit se envía al switch vecino o a la NIC destino. Se añade la latencia necesaria para transmitir el flit por el enlace (FLY ciclos). Por tanto, la latencia acumulada final será $INJ + SB + RT + AT + X + AT + X + SB + FLY$ ciclos en el peor de los casos, y sin contención, lo cual significa que el flit no tiene que esperar para usar los recursos del switch.

Es importante remarcar que se han tomado dos suposiciones: la latencia de almacenar en los buffers de salida incluye la de serialización y la latencia de almacenar en los buffers de entrada incluye la deserialización.

Finalmente, cada puerto tiene un ancho de banda de 100 Gbps y el tamaño de los flits es de 64 bits. Esto significa que la frecuencia de operación es de 1,6 GHz. Así, la latencia para cruzar el switch a través del Central Crossbar y sin contención es de 160 ciclos [4]. A partir de estos datos se ha realizado la siguiente distribución de latencias: 32 ciclos para el encaminamiento, 50 ciclos para el almacenamiento de flits en los buffers de entrada/salida, 16 ciclos para el asignador, 2 ciclos para cruzar los MPorts/Central XBar y 8 ciclos para la transmisión.

III. Soporte de OPA para calidad de servicio

La arquitectura Intel© Omni-Path© ofrece una serie de mecanismos orientados a proporcionar calidad de servicio [4] a las aplicaciones, flujos, paquetes, etc. Esos mecanismos son:

- Canales Virtuales(VLs): Proporcionan espacio dedicado para los paquetes. Además, los VLs se usan para evitar bloqueos en la red. OPA soporta 32 VLs.
- Canales de servicio (SCs): Diferencian paquetes de distintos SLs. El SC es el único identificador de QoS almacenado dentro de la cabecera del paquete. Cada SC se mapea a un solo VL. No obstante, un VL se puede compartir entre múltiples SCs. Los SCs se utilizan para evitar "deadlocks" debidos a la topología de red y para implementar técnicas de reducción del "head of line blocking" entre diferentes TCs. OPA soporta hasta 32 SCs, aunque, el SC15 está reservado para tareas de administración.
- Niveles de servicio (SLs): Son un grupo de SCs. Un SL puede tener múltiples SCs pero un SC solo pertenece a un SL. Los SLs se usan para separar paquetes de alta prioridad de paquetes de menos prioridad que pertenecen a la misma aplicación. OPA soporta hasta 32 SLs.
- Clases de tráfico(TCs): Representa un grupo de SLs orientados a distinguir el tráfico de una aplicación. Un TC puede tener múltiples SLs, pero un SL solo pertenece a un TC. OPA soporta hasta 32 TCs.
- vFabrics: Es un conjunto de puertos y uno o más protocolos de aplicaciones. Por cada vFabric, se aplica un conjunto de políticas de QoS. Un vFabric está asociado con un TC para QoS.

Los SLs se mapean a SCs a través de las tablas SL2SC y SC2SL, dependiendo de si los paquetes

son enviados o recibidos, respectivamente. Cada SC transporta tráfico de un solo SL en un solo TC. Las tablas SC2VL y VL2SC, determinan como los SCs se mapean a los VLs en cada puerto y viceversa. Un algoritmo configurable (*VLArbitration*) determina cómo los paquetes son planificados en los diferentes VLs. Además, se puede configurar el adelantamiento de paquetes permitiendo que paquetes de alta prioridad adelanten a paquetes de baja prioridad. Esto es, el algoritmo de arbitraje puede suspender el envío de un paquete de menor prioridad que se está transmitiendo para adelantar la transmisión de un paquete recién llegado al puerto de salida en una VL de alta prioridad. Cuando el paquete de alta prioridad es transmitido, el paquete suspendido reanuda su transmisión. El objetivo es reducir la latencia de los paquetes de alta prioridad.

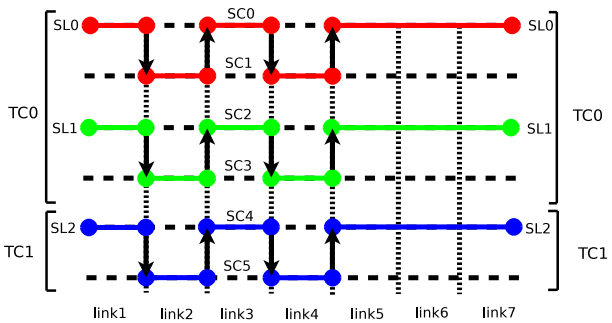


Fig. 3. Ejemplo de uso de TCs, SLs y SCs.

La figura 3 [4] muestra un ejemplo de uso de TCs, SLs y SCs a través de los caminos seguidos por los 3 flujos de tráfico (rojo, verde y azul) en una red OPA. Los diferentes enlaces cruzados por los paquetes se encuentran ordenados de 1 a 7. En este ejemplo, se asume el uso de 2 TCs (TC0 y TC1), 3 SLs (SL0, SL1 y SL2) y 6 SCs (SC0, SC1, SC2, SC3, SC4 y SC5). Cada SL tiene asignados 2 SCs, los cuales, a su vez, están asignados a 2 VLs. El TC0 (flujos en rojo y verde) podría usare para peticiones y respuestas de una biblioteca de comunicaciones PGAS[1]. El TC0 tiene asignado el SL0 (flujo rojo) y el SL1 (flujo verde), el SL0 es mapeado al SC0 y SC1 y el SL1 a SC2 y SC3. Por otro lado, el TC1 podría usarse para comunicaciones de almacenamiento en disco. El SL2 tiene asignados el SC4 y el SC5. El objetivo principal de asignar una pareja de SCs por SL es evitar bloqueos en la red, lo cual sucede típicamente en topologías como el toro. Por otro lado, los SLs del TC0 se usan para evitar bloqueos en el protocolo. Tal y como se puede observar en la figura 3, los paquetes pueden cambiar de SC de enlace a enlace; no obstante, el SL y el TC permanecen constantes de principio a fin [4].

[1] "Partitioned Global Address Space", son lenguajes de programación que combinan el paradigma de programación de memoria compartida con la localidad y rendimiento de paso de mensajes.

IV. Simple Bandwidth Table

Aprovechando los mecanismos explicados en la sección III, se ha implementado una primera técnica de calidad de servicio desarrollada en OPA, que hemos denominado Simple Bandwidth Table (SBT). SBT está basada en una tabla de arbitraje por puerto de salida con dos columnas: la primera muestra el SL y la segunda el peso asociado a ese SL. La tabla I muestra una configuración donde se proporciona una posible solución de calidad de servicio para el ejemplo mostrado en la figura 3. Se ha supuesto que el SL0 requiere de mucho ancho de banda y se le ha asignado un peso de 50, mientras que el SL1 y el SL2 necesitan un ancho de banda moderado y se les han asignado unos pesos de 30 y 20, respectivamente. Dichos pesos representan cuántos paquetes puede transmitir cada SL en cada puerto de salida. Cada vez que desde un puerto de salida se envía un paquete de un SL, se reduce en uno el peso asociado al SL. Por ejemplo, supongamos que el puerto 0 de un switch OPA tiene asociada una tabla de arbitraje como la mostrada en la tabla I. El puerto envía un paquete perteneciente al SL0, el peso restante de la entrada pasará a ser 49. Cuando el peso de todas las entradas de la tabla es igual a cero, SBT restablece los pesos. No obstante, existe una excepción, si un SL no tiene suficiente peso restante para enviar un paquete pero, es el único SL con algún paquete listo para la transmisión, SBT permite el envío. Esto evita la inanición de paquetes y desperdiciar ancho de banda. La división del ancho de banda se da como el ratio entre el peso total y el peso asignado a cada SL. Por ejemplo, en la tabla I, el peso total es 100, por lo que el SL0 obtiene el 50% del ancho de banda, el SL1 obtiene el 30% del ancho de banda y el SL2 obtiene el 20% del ancho de banda. Por motivos de simplicidad, se ha establecido que la suma de todas las entradas de las tablas SBT sea siempre 100. De esta forma, el ratio se puede calcular fácilmente.

TABLA I
Ejemplo de tabla STB.

SL	Peso
0	50
1	30
2	20

El algoritmo 1 muestra el funcionamiento genérico de SBT en cada puerto, donde la función *flitCabeza(vl)* permite extraer el flit en la cabeza de la cola un VL dado. Nótese que debido a que se usa la técnica *virtual cut-through*, este algoritmo solo se aplica al flit de cabecera de cada paquete. La tabla de arbitraje tiene una entrada por SL. Además, el SC es el único identificador incluido en el paquete [4], y por tanto, las tablas SC2SL son necesarias para obtener el SL a partir del VL y SC.

Nótese que cada puerto de salida implementa una tabla de arbitraje y, en esta técnica, la cantidad de entradas es igual al número de SLs presentes. Esta

Algorithm 1 Implementación genérica de SBT en OPA.

```
1:  procedure SBT(switch, pSalida)
2:      vl_con_flits_sin_peso ← maxVLs
3:      for i ← 0, NumVLs do
4:          vl ← (ultimo_vl_selec + i)%NumVLs
5:          if flits_VL_listo[i] && creditos_VL then
6:              flit ← flitCabeza(vl)
7:              flit_SL ← SC2SL[flit.SC]
8:              if t_arbitraje[pSalida][flit_SL] > 0
    then
9:                  ultimo_vl_seleccionado ← vl
10:                 t_arbitraje[pSalida][flit_SL] − −
11:                 return vl
12:             else
13:                 vl_con_flits_sin_peso ← vl
14:                 entradas_vacias + +
15:             end if
16:         end if
17:     end for
18:     if entradas_vacias = entradas_totales then
19:         restablecerTabla(switch, pSalida)
20:     end if
21:     if vl_con_flits_sin_peso ≠ maxVLs then
22:         ultimo_vl_selec ← vl_con_flits_sin_peso
23:         return vl_con_flits_sin_peso
24:     end if
25: end procedure
```

técnica de calidad de servicio junto con todos los mecanismos explicados en la sección III, se han implementado en OPASim. Además, se han implementado otros mecanismos necesarios para hacer compatible toda la implementación con el simulador y la arquitectura. Por último, se ha implementado un sistema capaz de forzar a los diferentes SLs a generar un porcentaje variable del ancho de banda. Este porcentaje se define al comienzo de la simulación.

V. Experimentos y resultados

Para probar el correcto funcionamiento del modelo de simulación presentado en la sección II y la técnica de calidad de servicio descrita en la sección IV, se han realizado una serie de experimentos usando OPASim. Dicha herramienta es capaz de ejecutar simulaciones utilizando diferentes tipos de tráfico, múltiples parámetros de configuración y extraer estadísticas de cada SL.

Para poder establecer una línea base de comparación, se ha usado un árbitro Round-Robin. Dicho algoritmo reparte de forma equitativa el ancho de banda entre todos los SLs y VLs presentes en la simulación. Por tanto, el árbitro Round-Robin representa el comportamiento del modelo de simulación cuando no se está usando ninguna técnica de calidad de servicio. Para estos experimentos, se ha usado un solo switch OPA, con una configuración que corresponde con la primera generación de switches OPA: 48 puertos, tamaño de flit de 64 bits, tamaños de cola de 256 flits, etc. (más información en [4]). Para cada simulación y punto mostrado en las figuras, se han ejecutado 30 simulaciones variando la semilla de generación de números aleatorios y calculado la media de los valores obtenidos.

Se han probado dos tipos de tráfico sintético: con contención y sin contención. Para el tráfico sintético con contención, se ha usado tráfico aleatorio uniforme. Para el tráfico sintético sin contención, una NIC con ID x envía paquetes a otra NIC y, tal que $y = (x+1)\%NICs$, donde $NICs$ es el número total de NICs en la simulación, en este caso 48.

Se han analizado diferentes configuraciones, modificando el número de SLs, SCs, VCs, ratio de generación de cada SL y tablas de arbitraje. Se han tomado en cuenta 4 métricas por SL: latencia nodo-a-nodo y latencia de la red, ambas medidas en nanosegundos; productividad media de la red, medida en flits/ciclo/NIC; y finalmente, productividad normalizada, es decir, la productividad total de cada SL con respecto a la productividad total del punto de simulación. Para facilitar la comprensión de las figuras, todos los SLs de la misma técnica de QoS se representan utilizando el mismo color, mientras que el mismo SL en técnicas diferentes de QoS se representa usando el mismo estilo de punto.

TABLA II
Configuración A

SL	SC	SC	VL	SL	%	SL	W
0	0	0	0	0	50	0	60
1	1	1	1	1	40	1	30
2	2	2	2	2	10	2	10

TABLA III
Configuración B

SL	SC	SC	VL	SL	%	SL	W
0	0 1	0	0	0	40	0	50
1	2	1	1	1	50	1	30
2	3	2	2	2	10	2	20
		3	2				

Las tablas II y III muestran las configuraciones de las tablas SL2SC, SC2VL, porcentaje de generación de mensajes y tabla de arbitraje de cada experimento. La tabla SL2SC (primera a la izquierda) incluye dos columnas, identificador de SL y SCs asociados a ese SL. La tabla SC2VL tiene dos columnas: identificador del SC y el VL asociado. La tabla representada por la dupla SL y % es la tabla de porcentaje de generación por SL, indicando el identificador del SL y el porcentaje del total del tráfico que el SL generará en cada punto de simulación. Por ejemplo, en un momento determinado de la simulación se está inyectando a un ratio de 1 flit/ciclo/NIC a la red, los SLs 0, 1 y 2 de la configuración mostrada en la tabla II generarán 0,5 flits/ciclo/NIC, 0,4 flits/ciclo/NIC y 0,1 flits/ciclo/NIC, respectivamente. Por último, la tabla de arbitraje representada por la pareja SL y W, muestra el identificador de cada SL y el peso asociado a este SL.

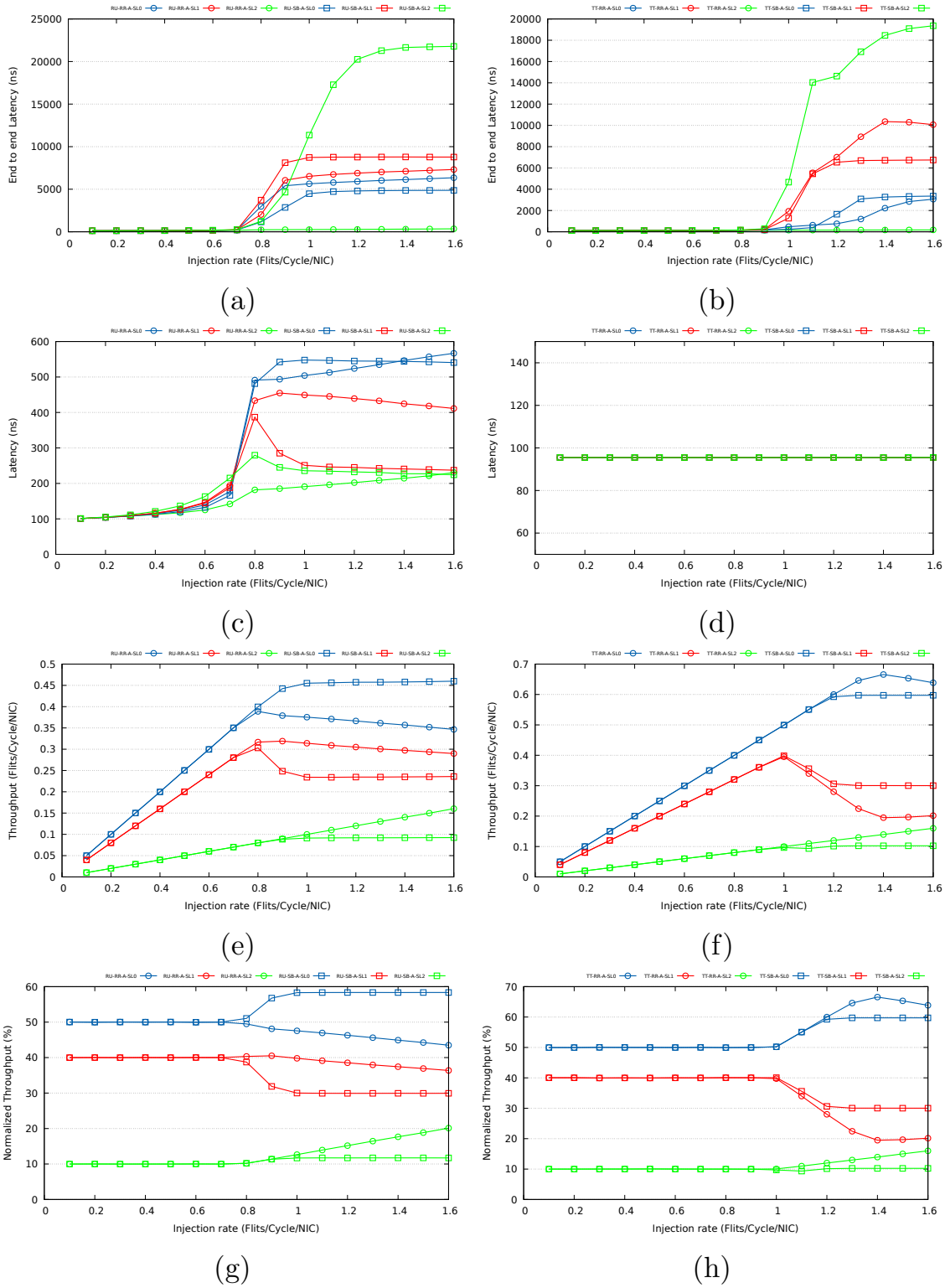


Fig. 4. Rendimiento por SL de SBT con la configuración mostrada en la tabla II contra un árbitro Round-Robin (RR) usando *tráfico uniforme* en (a), (c), (e) y (g) y *tráfico sin contención* en (b), (d), (f) y (h) como carga de red.

Las figuras 4 y 5 muestran los resultados obtenidos por dichas configuraciones. En cada una de las figuras se muestra: latencia nodo-a-nodo (a,b), latencia de la red (c,d), productividad (e,f) y productividad normalizada (g,h) por SL. Estas métricas se representan en los términos descritos anteriormente.

En las figuras puede observarse que SBT no asegura cumplir requisitos de latencia, tan solo de ancho de banda. Cuando no existe contención, la latencia nodo-a-nodo (a,b) presenta la misma tendencia que en el escenario con contención, aunque con unos valores inferiores. Cuando no existe contención, los paquetes no sufren ninguna demora adicional en la red, por lo que podrán usar los recursos del switch tan pronto como los soliciten. Este hecho se puede confirmar con la latencia de los paquetes en la red

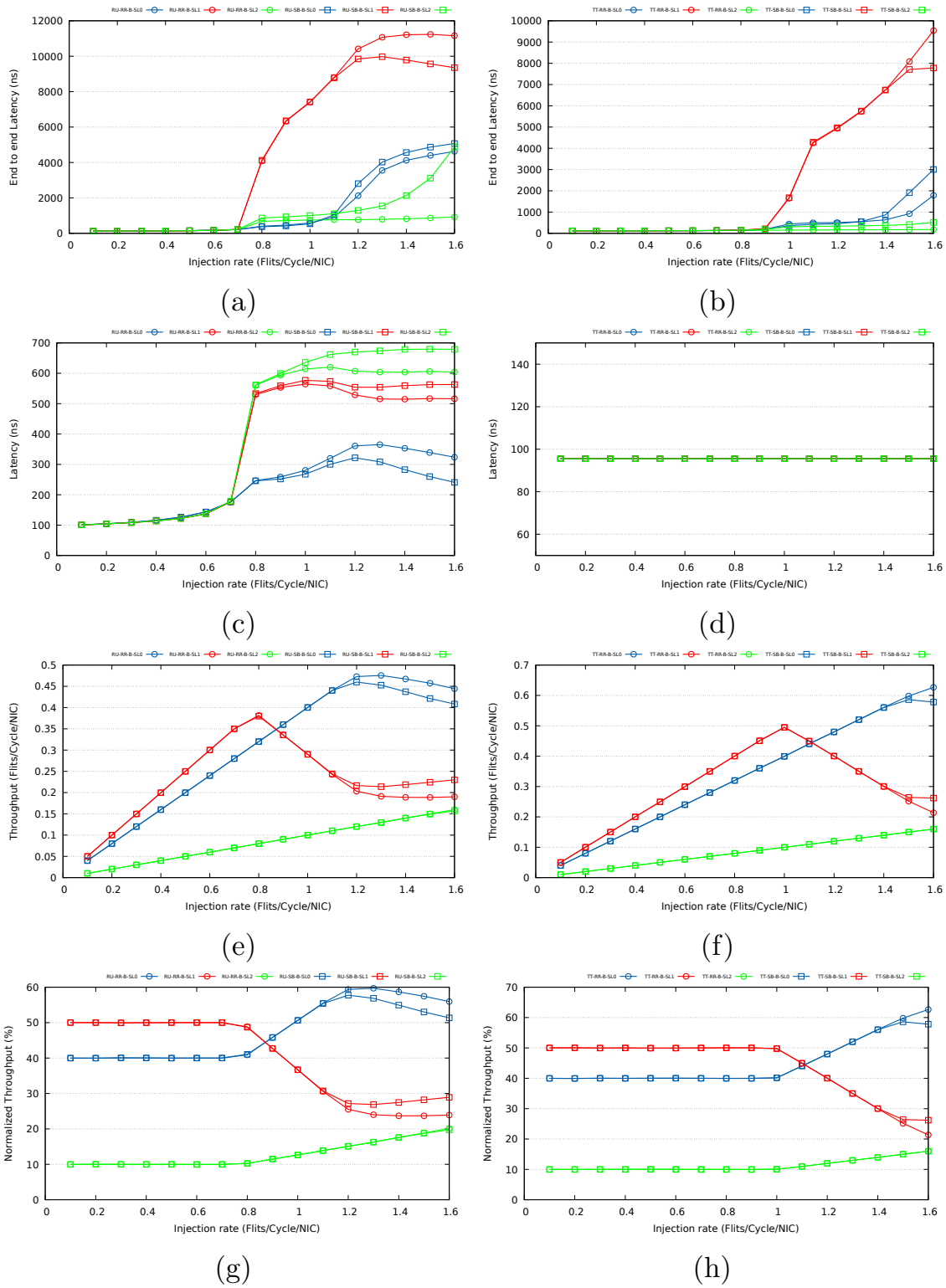


Fig. 5. Rendimiento por SL de SBT con la configuración mostrada en la tabla III contra un árbitro Round-Robin (RR) usando *tráfico uniforme* en (a), (c), (e) y (g) y *tráfico sin contención* en (b), (d), (f) y (h) como carga de red.

(c,d), donde la latencia en el escenario sin contención permanece constante. La única restricción que se impone a los paquetes cuando no hay contención es la tasa de generación y el reparto del ancho de banda por SL. Por ejemplo, en la configuración A, el SL0 tiene las latencias nodo-a-nodo más bajas, mientras que el SL1 tiene las más altas, ya que, aunque SL0 genera mayor cantidad de tráfico que SL1, tiene un mayor ancho de banda reservado, por lo que la latencia nodo-a-nodo se ve reducida. Por otro lado, el SL2 tiene el menor ancho de banda reservado, pero también una tasa de generación menor que el SL1 y, por tanto, un número de paquetes totales menor para transmitir. Esto implica una menor latencia nodo-a-nodo que la obtenida por el SL1.

En lo que respecta a la productividad, en el caso del tráfico con contención la productividad no alcanza aparentemente los requisitos impuestos por las tablas QoS. Esto se debe a que el switch entra en un estado de saturación (a partir del ratio de inyección 0,7 flits/ciclo/NIC), es decir, recibe más tráfico del que es capaz de procesar y entregar, antes de alcanzar una productividad teórica máxima. Por ejemplo, en la configuración A, se ha definido un 60% del ancho de banda para el SL0, pero tan solo obtiene 0,45 flits/ciclo/NIC. No obstante, esto significa un 60% de la productividad conseguida por el switch bajo un estado de saturación tal y como se ha configurado en la tabla de arbitraje. Este hecho se puede contrastar en las figuras de productividad normalizada (g,h). En el escenario sin contención, la productividad alcanzada por cada uno de los SLs, es prácticamente la misma que la configurada en las tablas de arbitraje.

Nótese que en lo referente a SBT, ambas configuraciones penalizan al SL1 cuando no hay suficiente ancho de banda para satisfacer los requisitos de QoS de los SLs 0 y 2. Mientras hay suficiente ancho de banda sin usar, el SL1 puede obtener el 40% de la productividad ofrecida. No obstante, después de este punto, el ancho de banda no es suficiente para enviar los paquetes generados por el SL0. Debido a que el SL2 ha alcanzado su ancho de banda máximo garantizado, el ancho de banda utilizado anteriormente por el SL1 ahora es usado por el SL0 para satisfacer sus requisitos. Por lo tanto, la productividad del SL1 disminuye hasta el 30%, acorde con la configuración de las tablas de arbitraje. Así, se puede asegurar que SBT funciona correctamente según lo esperado.

VI. Conclusiones

Este artículo describe OPASim, un simulador dirigido por eventos en el cual se implementa un modelo de simulación basado en la arquitectura Intel© Omni-Path©.

Además de los mecanismos básicos para el trasiego de los paquetes de origen a destino, se han implementado las características de OPA encargadas de proporcionar calidad de servicio a paquetes y aplicaciones. Entre estos mecanismos se pueden destacar: las tablas SL2SC, SC2VL, las tablas de arbitraje y los elementos secundarios para probar y comparar el simulador y los resultados obtenidos.

Se ha probado y comparado el algoritmo de calidad de servicio SBT contra el algoritmo Round-Robin. Los experimentos realizados, prueban que el algoritmo de QoS SBT, es capaz de segregar tráfico por SL y proporcionar garantías en ancho de banda con respecto a la línea base de comparación establecida, Round-Robin. Se ha podido comprobar, además, que los diferentes SLs obtienen bajo condiciones de saturación (periodos en los cuales la red de interconexión recibe más paquetes de los que puede procesar) el ancho de banda estipulado. SBT es, por tanto, un algoritmo bien balanceado. La productividad final se encuentra muy cerca de lo esperado, habiendo una pequeña fluctuación despreciable en condiciones de saturación.

Como trabajo futuro, se pretende implementar más técnicas de calidad de servicio adaptadas a OPA. Por otra parte, se pretende implementar técnicas de QoS pensadas para sacar el mayor partido a los mecanismos disponibles. Dichas técnicas asegurarán requisitos de latencia a los diferentes SLs. También se planea implementar en OPASim tablas de adelantamiento. Estas tablas permiten que paquetes de alta prioridad suspendan el envío de un paquete de baja prioridad en curso, para enviar aquellos con mayor prioridad. Con la inclusión de esta característica y su adaptación a SBT, OPASim será capaz de asegurar requisitos de ancho de banda y latencia.

AGRADECIMIENTOS

Este trabajo ha sido financiado por la Junta de Comunidades de Castilla-La Mancha, la Comisión Europea (fondos FEDER) bajo el proyecto SBPLY/17/180501/000498 y el Ministerio de Ciencia, Innovación y Universidades bajo el proyecto RTI2018-098156-B-C52. También está cofinanciado por la Universidad de Castilla-La Mancha y FEDER bajo el marco del proyecto 2019-GRIN-27060. Javier Cano-Cano es financiado por el Ministerio de Economía y Competitividad bajo la beca FPI BES-2016-078800.

Referencias

[1] Michael Feldman and Addison Snell, "A new high performance fabric for HPC," *Intersect360 Research white paper, May*, 2016.

[2] Gregory F. Pfister, "An introduction to the InfiniBand architecture," *High Performance Mass Storage and Parallel I/O*, vol. 42, no. 1, pp. 617–632, 2001.

[3] Francisco J. Andújar, Juan A. Villar, Jose L. Sánchez, Francisco J. Alfaro, and Jesús Escudero-Sahuquillo, "An open-source family of tools to reproduce MPI-based workloads in interconnection network simulators," *The Journal of Supercomputing*, vol. 72, no. 12, pp. 4601–4628, 2016.

[4] Mark S. Birrittella, Mark Debbage, Ram Huggahalli, James Kunz, Tom Lovett, Todd Rimmer, Keith D. Underwood, and Robert C. Zak, "Intel® Omni-Path architecture: Enabling scalable, high performance fabrics," in *High-Performance Interconnects (HOTI), 2015 IEEE 23rd Annual Symposium on*. IEEE, 2015, pp. 1–9.

[5] Mark S. Birrittella, Mark Debbage, Ram Huggahalli, James Kunz, Tom Lovett, Todd Rimmer, Keith D. Underwood, and Robert C. Zak, "Enabling scalable high-performance systems with the Intel Omni-Path architecture," *IEEE Micro*, vol. 36, no. 4, pp. 38–47, 2016.

[6] Parviz Kermani and Leonard Kleinrock, "Virtual cut-through: A new computer communication switching technique," *Computer Networks (1976)*, vol. 3, no. 4, pp. 267 – 286, 1979.

[7] Madhavapeddi Shreedhar and George Varghese, "Efficient fair queueing using deficit Round Robin," in *ACM SIGCOMM Computer Communication Review*. ACM, 1995, vol. 25, pp. 231–242.

[8] Salil S. Kanhere, Harish Sethu, and Alpa B. Parekh, "Fair and efficient packet scheduling using elastic Round Robin," *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, no. 3, pp. 324–336, 2002.

[9] Yuval Tamir and Gregory L. Frazier, "Dynamically-allocated multi-queue buffers for VLSI communication switches," *IEEE Transactions on Computers*, vol. 41, no. 6, pp. 725–737, 1992.

Modelado de Cargas de Trabajo de Aplicaciones en Herramientas de Simulación de Redes Data-Center

Luis Gonzalez-Naharro [1], Jesus Escudero-Sahuquillo [2], Pedro J. García [3], Francisco J. Quiles, [4]
Jose Duato [5], Wenhao Sun [6], Xiang Yu [7], y Hewen Zheng [8]

Resumen— **Los Data-Centers son utilizados comúnmente por los proveedores de cloud más importantes del mundo para proporcionar recursos de almacenamiento y computación y, en base a estos recursos, servicios y aplicaciones de Tecnologías de la Información avanzados. Con la explosión de datos esperada en los próximos años, los centros de datos requerirán nuevas arquitecturas para hacer frente a los nuevos requisitos de las aplicaciones y los usuarios. Uno de los subsistemas cruciales dentro de la arquitectura del centro de datos que debe evolucionar de acuerdo con los nuevos requisitos es la red de interconexión o la red del centro de datos (DCN). El rendimiento de la DCN (básicamente, alto ancho de banda de comunicación y baja latencia) debe estar garantizado, de lo contrario, la DCN se convertirá en el cuello de botella del sistema. Hay varios problemas clave en los que los diseñadores de la DCN deben tomar decisiones, como la topología de la red, el algoritmo de enrutamiento, el control de la congestión, etc. Un aspecto importante que impacta en el diseño de la DCN son los patrones de comunicación de red generados por las aplicaciones y los servicios. En ese sentido, un modelado preciso de estas cargas de trabajo ayudaría a los diseñadores de redes a tomar mejores decisiones. En este artículo, presentamos un análisis de los pocos estudios disponibles sobre el modelado de tráfico para DCN, con el fin de recopilar un conjunto de parámetros que definen el comportamiento de las cargas de trabajo de tráfico comunes. Basándonos en estos parámetros, hemos implementado un generador de tráfico de DCN sintético, que se ha incluido en nuestro marco de simulación para alimentar a la red con las cargas de tráfico inferidas. Hemos realizado simulaciones exhaustivas para probar el impacto de la variación de los parámetros en el rendimiento de la red. De los resultados obtenidos, podemos concluir que la distribución de los destinos es crucial para el rendimiento de la red. Un acceso frecuente a ciertos destinos genera escenarios incast que conducen a situaciones de congestión y HoL blocking, afectando a otros flujos que no contribuyen a la situación incast y deteriorando el rendimiento de la red.**

Palabras clave— **Redes Data-Center, caracterización de carga de trabajo, modelado de carga de trabajo, comunicación de red, simulación.**

[1]Departamento de Sistemas Informáticos, UCLM, España: luis.gnaharro@uclm.es
[2]Departamento de Sistemas Informáticos, UCLM, España: jesus.escudero@uclm.es
[3]Departamento de Sistemas Informáticos, UCLM, España: pedrojavier.garcia@uclm.es
[4]Departamento de Sistemas Informáticos, UCLM, España: francisco.quiles@uclm.es
[5]Universitat Politècnica de València, España: jduato@disca.upv.es
[6]Huawei Technologies Co., Ltd., China: sam.sunwenhao@huawei.com
[7]Huawei Technologies Co., Ltd., China: yolanda.yu@huawei.com
[8]Huawei Technologies Co., Ltd., China: zhenghewen@huawei.com

I. Motivación

Un Data-Center (DC) [1] (también conocido como Hyperscale DC o Hyperscaler) es un conjunto de recursos de computación y almacenamiento agrupados que usan redes de comunicación para alojar aplicaciones y almacenar datos. En los últimos años, las plataformas de cloud más grandes del mundo, como Microsoft, Google, Facebook y Amazon, están actualizando la capacidad de almacenamiento y computación de sus DC para soportar el crecimiento de la cantidad de espacio de almacenamiento y la potencia computacional requeridas. El resultado de estos esfuerzos de inversión de diversas empresas y compañías, es que la cantidad de Hyperscale DC aumentará significativamente para el año 2020. En este contexto, los DC requieren innovaciones en su infraestructura de hardware y software para hacer frente a las demandas de las aplicaciones cada vez más complejas y las demandas de los usuarios en tiempo real.

Una parte crítica de la arquitectura de los DC es la red de interconexión que se encarga de comunicar los nodos de servidores de computación y almacenamiento. De hecho, el número creciente de servidores interconectados exige que la arquitectura de los DC sea tolerante a fallos, rentable y escalable, con alto ancho de banda de comunicación entre nodos y bajo tiempo de respuesta (latencia) en la transmisión de datos. Como resultado, la DCN debe diseñarse a conciencia; de lo contrario, se convertirá en el cuello de botella del sistema [2]. Específicamente, las DCN están compuestas por tarjetas de interfaz de red (NIC), conmutadores, bridges y cableado. Las NIC generalmente se conectan en los nodos del servidor al subsistema PCIe, conectándolos directamente a la CPU y la memoria. Los conmutadores y bridges interconectan los servidores de DC utilizando cables, creando topologías de red específicas, como la topología CLOS sin bloqueo, que se usa en muchas instalaciones de DC [3], ya que proporciona una gran diversidad de rutas y facilita el uso de algoritmos de enrutamiento de balanceo eficiente de carga.

Básicamente, los DCN deben ofrecer un alto ancho de banda de comunicación y baja latencia, y su diseño debe tener en cuenta estos requisitos de rendimiento. Específicamente, hay varios proble-

mas de diseño claves que afectan directamente al rendimiento de la red, como la topología de red configurada, el algoritmo de enrutamiento, los mecanismos de control de congestión implementados (si los hay), la provisión (o no) de la calidad del servicio, etc.

Se debe tener en cuenta que las cargas de tráfico de las aplicaciones que se ejecutarán en el DC también determinan el rendimiento del DCN y tienen una fuerte influencia en los problemas de diseño de red mencionados anteriormente.

Más precisamente, los DCN se pueden estresar con al menos cuatro casos de uso críticos o cargas de trabajo [4]:

- Los servicios On-Line Data Intensive, los cuales devuelven respuestas inmediatas a las peticiones de los usuarios que llegan al sistema DC a un alto ritmo. La respuesta del sistema DC deber ser lo suficientemente rápida como para que el usuario final no perciba el retardo de procesamiento.
- Aplicaciones de deep learning, que aprenden a través de un proceso llamado "entrenamiento" para manejar cada nuevo problema usando la misma red neuronal profunda (DNN, por sus siglas en inglés). El entrenamiento de una DNN es una aplicación altamente paralela que requiere dispositivos informáticos especializados para proporcionar un alto rendimiento y una baja latencia. Sin embargo, el tiempo de comunicación involucrado en el entrenamiento puede superar a las ganancias de agregar un mayor número de dispositivos informáticos.
- Protocolos de comunicación de almacenamiento distribuido. La aparición de nuevos requisitos de almacenamiento en la esfera de datos global plantea nuevos desafíos para el subsistema de almacenamiento de los DCs, centrados en el desarrollo de protocolos de comunicación avanzados para aumentar la tasa de ancho de banda de acceso.
- Cloudificación de redes empresariales. Las oficinas centrales (COs por sus siglas en inglés) de las telecomunicaciones modernas están adaptando su infraestructura a la nube, migrando servicios y plataformas tradicionales, como VPNs, telefonía IP o almacenamiento a sistemas DC. Si bien las redes de CO tradicionales implementaban un conjunto de dispositivos dedicados para funciones especiales, carecían de la flexibilidad para escalar y adaptarse a cambios futuros. En consecuencia, las CO modernas están adaptando su arquitectura, construyendo sistemas DC potentes que utilizan redes de alto rendimiento y sin pérdidas.

De los cuatro casos de uso críticos descritos anteriormente, podemos afirmar que hay una gran cantidad de datos para procesar y administrar, que los usuarios finales deben recibir en una fracción imperceptible de tiempo. La administración de datos es un objetivo compartido con las aplicaciones tradicionales de clústeres de computación de alto rendimiento (HPC), donde las aplicaciones están optimizadas para minimizar el tráfico de datos en la red. Sin embargo, en el entorno de los DCs, es más complicado optimizar estas aplicaciones debido a dos razones principales: el comportamiento de la aplicación y la carga generada son significativamente variables, y la carga de trabajo en los DCs depende de la cantidad de solicitudes que realicen los usuarios que interactúan con los servicios en todo el mundo. Sin embargo, como sucede con los sistemas HPC, si pudiéramos predecir el comportamiento de estas interacciones, sería más fácil diseñar DCNs capaces de manejar eficientemente las cargas de trabajo de las aplicaciones. De hecho, el modelado de las comunicaciones de red realizadas por los servidores que ejecutan esas cargas de trabajo permitiría utilizar estos modelos para alimentar las herramientas de simulación, a fin de probar y verificar nuevos diseños de red destinados a mejorar el rendimiento de la red. Por lo tanto, el modelado de las cargas de trabajo de DCs es crucial para el diseño de mejores arquitecturas de DCN.

Desafortunadamente, los operadores de DCs generalmente son reticentes a compartir los requisitos reales de sus aplicaciones. Por lo tanto, se sabe muy poco acerca de las características del tráfico a nivel de red de los centros de datos actuales, excepto por algunos estudios de Facebook [5] y Microsoft [6]. Estos y otros estudios [7] nos han brindado algunas ideas sobre cómo modelar las cargas de trabajo de las comunicaciones en las DCN. Este artículo presenta una investigación realizada para estudiar y modelar los patrones de comunicación generados en las DCNs por aplicaciones reales. Básicamente, hemos analizado los estudios mencionados anteriormente de Microsoft y Facebook y hemos inferido un modelo basado en los parámetros proporcionados en estos estudios, como el número de flujos generados, los instantes en que se generan estos flujos, los tamaños de estos flujos, los tamaños de los paquetes y la distribución de destinos (por ejemplo, Normal, Uniforme, Poisson, etc.). Además, hemos implementado este modelo en un generador de tráfico de red sintético, que hemos incluido en nuestro programa de simulación de redes habitual. Para probar nuestra implementación, hemos realizado simulaciones para evaluar cómo la variación de los parámetros del modelo tiene un impacto en las cargas de trabajo de las aplicaciones y en el rendimiento de la red. A partir de los resultados obtenidos, hemos observado que la distribución de destinos influye de manera significativa en el rendimiento de la red, ya que el sobre-uso de destinos no solo disminuye el rendimiento general, sino que también crea situaciones incast que conducen a la congestión y sus problemas relacionados [8], como el bloqueo de cabeza de línea (HoL blocking). En concreto, una situación incast aparece cuando diferentes flujos de tráfico convergen en un único puerto de salida dentro de un switch, generando una situación de

congestión. Tal y como explicaremos a lo largo del artículo, los flujos de tráfico que no contribuyen a la situación de incast pueden verse afectados por los efectos del HoL blocking, por lo que el rendimiento de la red se degrada considerablemente.

El resto de este artículo está organizado de la siguiente manera. La sección II proporciona antecedentes relacionados con DCNs, cargas de trabajo de aplicaciones y herramientas de simulación que modelan DCNs. La sección III proporciona información técnica sobre cómo hemos caracterizado y modelado el generador de tráfico, y cómo se ha implementado e integrado en nuestro programa de simulación. La sección IV muestra los resultados obtenidos de los experimentos para probar y verificar nuestro modelo de tráfico utilizando varias distribuciones de destinos. Finalmente, en la sección V se extraen algunas conclusiones.

II. Antecedentes

A. Redes Data-Center

Como se mencionó anteriormente, las aplicaciones y servicios modernos demandan un alto rendimiento y una latencia ultra bajas a la DCN, ya que deben ejecutarse con una baja sobrecarga de CPU. Entre las tecnologías de DCN, Ethernet es la opción principal para las comunicaciones de servidor a servidor en la nube y los DCs de los proveedores de servicios. La Ethernet Alliance (EA) apoya el desarrollo de estándares que definen la tecnología Ethernet gracias a los grupos de trabajo o subcomités que realizan esfuerzos en torno a iniciativas de estandarización específicas, como el Instituto de Ingenieros Eléctricos y Electrónicos (IEEE) y el Grupo de Trabajo de Ingeniería de Internet (IETF).

De hecho, IEEE e IETF están trabajando hoy en día en nuevos métodos de señalización, factores de forma y fibra monomodo y multimodo a velocidades de 100, 200 y 400 Gbps, entre otras actividades. Por lo tanto, el ancho de banda de Ethernet continúa creciendo exponencialmente, y se espera que alcance velocidades de Terabits (tecnología TbE) en algún momento entre 2020 y 2030, según la Hoja de ruta de la Ethernet Alliance.

Sin embargo, el uso de la pila TCP/IP introduce una sobrecarga de CPU muy alta, lo cual es indeseable para los proveedores de la nube que monetizan el uso de la CPU, ya que es problemático medir la sobrecarga de la CPU.

Para evitar la latencia adicional introducida por los protocolos de la pila TCP/IP y la sobrecarga de la CPU, existen soluciones, como el acceso directo a memoria remoto (RDMA) sobre Ethernet convergente (RoCE). RoCE supera los problemas anteriores al reemplazar las capas de transporte Ethernet tradicionales al tiempo que conserva el uso de las capas físicas (PHY) y de acceso al medio (MAC) de Ethernet.

Otras tecnologías, como iWarp o Fibre Channel sobre Ethernet convergente (FCoE) también tratan los problemas de latencia introducidos por la retransmisión.

Con el fin de satisfacer las demandas de rendimiento de aplicaciones y servicios específicos, muchas empresas han construido sus propias redes y DCs. Por ejemplo, Facebook ha publicado algunos detalles sobre su DC y las cargas de trabajo [5]. Específicamente, los emplazamientos de los DCs de Facebook se dividen en edificios de DCs, que a su vez están compuestos por varios clústeres. Estos clústeres están formados por un conjunto de máquinas conectadas a conmutadores top-of-rack (RSW), y cada uno de estos RSW está conectado a cuatro conmutadores de clúster (CSW). Finalmente, los CSW se conectan entre sí mediante conmutadores de agregación llamados Fat Cats (FC).

Con respecto a los servidores más comunes, hay cuatro tipos en los que se centra este estudio [5]: servidores web, que manejan solicitudes web; *cache followers*, que sirven solicitudes de caché; *cache leaders*, que mantienen la consistencia del caché con las bases de datos principales; y servidores Hadoop, que realizan análisis de minería de datos fuera de la red principal.

B. Cargas de Trabajo de Aplicaciones en DCs

Recientemente, ha habido un interés creciente en modelar los patrones de tráfico de los DCs, especialmente los que están presentes en los DCs de Facebook. Los estudios realizados en algunos de estos DCs de Facebook muestran que el comportamiento de su tráfico difiere del presente en otros DCs: el tráfico no es local a los racks y no es todos-a-todos, la mayoría de los flujos son de larga duración pero no son grandes, y la mediana del tamaño de paquete es muy pequeña (menos de 200 bytes, con la excepción del tráfico de Hadoop). El estudio en [5] en particular se centra en analizar el tráfico en servidores web, servidores de caché (tanto *followers*, que sirven a los servidores web; y *leaders*, que mantienen la coherencia con las bases de datos) y servidores Hadoop, y muestra que, si bien el tráfico de Hadoop es similar al reportado en la literatura, el tráfico para los otros servicios no sigue esta tendencia (no hay localidad a nivel de rack, como ya se ha dicho). Este estudio también analiza la distribución de los destinos a lo largo del tiempo para el tráfico de estos servicios. El tráfico de los servidores de Hadoop se mantiene principalmente dentro de los racks y clusters debido a que el análisis de minería de datos se realiza de manera offline. Estas cargas de trabajo son muy inestables a lo largo del tiempo debido a la alternancia entre las fases de computación y comunicación del análisis. El tráfico de los servidores web se encuentra principalmente dentro del clúster, al igual que el tráfico de los *cache followers* (ya que sirven a los servidores web). El tráfico de los *cache leaders* se dirige principalmente a otras máquinas dentro del mismo centro de datos, y también fuera del centro de datos (ya que deben mantener la consistencia de la caché). Finalmente, este mismo estudio [5] también muestra que, aunque el tráfico es bastante estable

en las escalas de tiempo de segundo y de día, en la escala sub-segundo, existen algunas irregularidades causadas por los llamados *heavy hitters* (el conjunto mínimo de flujos, hosts o racks que son responsables del 50 % del volumen de tráfico observado, en bytes, durante un período de tiempo fijo). Sin embargo, la solución de estas irregularidades no es viable desde el punto de vista del estudio mencionado, dada la dificultad de detección de estos *heavy hitters* y a la falta de beneficios de solucionarlas.

Por lo tanto, creemos que proponer un nuevo modelo de patrón de comunicación de tráfico para analizar dichas irregularidades puede ayudar a comprender mejor las situaciones incast a las que pueden conducir estos *heavy hitters* (ya que tener una pequeña cantidad de nodos que generan grandes ráfagas de tráfico puede causar congestión) . Se debe tener en cuenta que muchas referencias utilizan estrategias de balanceo de carga para resolver el problema de incast[9]. Sin embargo, analizar dichas soluciones está fuera del ámbito de este documento, donde solo modelamos varios tipos de patrones de comunicación basados en modelos de distribución de destinos. Los modelos de tráfico propuestos se basan en ciertas distribuciones estadísticas habituales, las cuales se describen en la siguiente subsección.

C. Modelos de patrones de comunicación

Los patrones de comunicación que se utilizan en este estudio se centran principalmente en el modelado de los llamados *modelos de distribución de destinos*: es decir, determinamos los nodos de destino para los mensajes generados en los nodos finales de la red. Aunque es obvio que, en redes reales, un nodo final no decidirá a dónde enviar un mensaje basándose en la selección de un valor aleatorio dada una distribución, en realidad estos modelos ayudan a comprender un aspecto clave: cómo varía el rendimiento de la red dependiendo de si los destinos están distribuidos uniformemente o sesgados hacia ciertos nodos. De hecho, los resultados presentados en la Sección IV muestran que en este último caso, el rendimiento disminuye debido a las situaciones incast.

Los modelos de patrón de comunicación (es decir, modelos de distribución de destinos) de los flujos de tráfico generados se pueden clasificar en los siguientes modelos estadísticos:

- *Distribución uniforme.* Este modelo es el más sencillo. Básicamente, todos los nodos finales tienen la misma probabilidad de ser elegidos como destino objetivo. En este caso, los escenarios incast rara vez aparecen, ya que todos los nodos finales de la red reciben la misma cantidad de flujos de tráfico.
- *Distribución normal.* Este modelo utiliza una distribución normal, con $\mu = num\_nodes/2$ y $\sigma = num\_nodes/6$. Se debe tener en cuenta que μ es la media o valor esperado de la distribución y también su mediana y moda, mientras que σ es la desviación estándar. Estas fórmulas garantizan que la probabilidad de elegir un nodo válido (que va de 0 a num_nodes - 1) es 99.73 %. Todos los valores no válidos son corregidos automáticamente por nuestra herramienta de simulación utilizando un operador de módulo. Como se puede inferir, este modelo conducirá a situaciones de incast ligeras, ya que los nodos con ID $num\_nodes/2$ tendrán más probabilidades de ser seleccionados como destinos que otros.
- *Distribución de Poisson.* Finalmente, el tercer modelo utilizado es una distribución de Poisson, con $\mu = num\_nodes/2$. Este es el modelo que genera más incast en comparación con los demás, ya que solo un pequeño rango de nodos se consideran como destinos de todos los mensajes generados. De hecho, en el caso de una red de 128 nodos, solo los nodos que oscilan entre 50 y 78 tienen una probabilidad de ser elegidos de más del 1 %.

III. Descripción del Modelado de las Cargas de Trabajo

En esta sección, describimos nuestro modelo de caracterización de cargas de trabajo, y cómo lo hemos implementado en nuestro simulador.

A. Caracterización de las Cargas de Trabajo

Básicamente, asumimos que una carga de trabajo es un conjunto de flujos de tráfico que se pueden generar en un período de tiempo. Basándonos en la aplicación o servicio que genera un flujo dado, podemos asociar diferentes flujos a diferentes clases de tráfico, lo que también puede generar flujos de diferentes tamaños en un instante de tiempo diferente. Además, los nodos finales de origen y destino de estos flujos pueden ser diferentes según la aplicación o el servicio (es decir, la clase de tráfico). Por lo tanto, cada clase de tráfico tiene un patrón de comunicación diferente que puede correlacionarse con una distribución específica de destinos.

Basándose en estas ideas, el primer paso para caracterizar las cargas de trabajo de tráfico es definir un conjunto de parámetros que identifiquen los patrones de tráfico con un nivel aceptable de precisión. Sobre la base del tráfico de red observado en algunos de los DCs de Facebook, hemos definido un conjunto de parámetros que son comunes a las cargas de trabajo utilizadas en estos sistemas. Estos parámetros son los siguientes:

- `number_flows`. Este parámetro define el número total de flujos que se generarán en la carga de trabajo.
- `Number_of_classes`. Este parámetro especifica el número de clases de tráfico que se generarán.
- `traffic_class`. Este parámetro identifica cada clase de tráfico. Específicamente, para cada clase de tráfico debemos especificar una ID interna, el tamaño de los flujos de tráfico que pertenecen a esa clase (o un rango que define los tamaños de los flujos más pequeños y

más grandes que pueden estar en esta clase de tráfico) y el porcentaje de flujos totales que pertenecen a esta clase de tráfico. Por medio de estos parámetros, podemos modelar flujos pequeños (es decir, *mices*) o flujos largos (es decir, *elephants*).

- `max_gen_time`. Define el tiempo máximo hasta el cual todos los flujos de tráfico deben haberse generado en el sistema. Se debe tener en cuenta que asumimos que el DCN puede absorber todo el tráfico generado o tiene al menos búferes para almacenar el tráfico generado donde los flujos generados bloqueados pueden esperar hasta que se inyecten en la red. También se debe tener en cuenta que si la NIC del nodo final de origen está bloqueada (es decir, no tiene espacio para almacenar un flujo de tráfico específico ya generado), la generación del flujo se pospondrá hasta que haya espacio.
- `destination_dist`. Especifica, para cada flujo, la función de densidad de probabilidad del destino de ese flujo. Se debe tener en cuenta que esta función determina el sobre-uso de cada patrón de tráfico y determina si el tráfico generará escenarios incast en la red. Actualmente, solo están implementadas distribuciones uniformes, Normal y Poisson.

Se debe tener en cuenta que este modelo requiere analizar la distribución de los nodos finales de destino generados por cada nodo final de origen. Por lo tanto, se puede ampliar siempre que definamos nuevas distribuciones de destinos. En la siguiente subsección, explicamos cómo se ha implementado este modelo de caracterización de la carga de trabajo en nuestro modelo de simulación.

B. Implementación del Modelo

Nuestro modelo de tráfico ha sido implementado en el lenguaje C++, ya que lo hemos integrado en nuestro programa de simulación, como describimos en la Sección IV. La figura 1 muestra el diagrama UML que describe las dos clases que hemos utilizado para implementar este modelo.

Específicamente, para generar los flujos a simular, nuestro simulador tiene una clase llamada `DataCenterApplication`, que a su vez extiende de otra clase llamada `ApplicationManager`. Juntas, estas dos clases proporcionan la funcionalidad requerida para generar y monitorizar todos los flujos. Primero, el simulador le da a cada flujo un tiempo de inicio aleatorio en un nodo de origen aleatorio, dentro del intervalo de tiempo dado por el parámetro `max_gen_time`. Cuando el tiempo de simulación alcanza este tiempo de inicio aleatorio, genera el flujo correspondiente, con un tamaño dado en el intervalo dado en la información de clase de tráfico. Se debe tener en cuenta que el tamaño del flujo se genera **aleatoriamente** dentro de ese intervalo. Sin embargo, si la tarjeta de interfaz de red (NIC) del nodo de origen donde se debe generar el flujo está bloqueada (seguramente porque sus colas internas estén llenas), el flujo se pondrá en espera hasta que se desbloquee la NIC. Si la NIC tiene suficiente espacio para almacenar el flujo, entonces se genera correctamente, dándole un ID, y un destino de acuerdo con el esquema de generación de destinos. Se deber tener también en cuenta que, dependiendo de la función de distribución (por ejemplo, Uniforme, Normal o Poisson), las ocurrencias de los destinos elegidos por cada flujo variarán. Por ejemplo, si se elige la distribución de Poisson, entonces habrá un destino visitado con mayor frecuencia por los flujos de tráfico (es decir, tendremos nodos finales sobre-utilizados).

Además, para obtener métricas sobre la generación de flujos y tiempos de finalización, inicializamos una entrada de tabla hash para cada generación de flujo de tráfico, dándole como valor el tiempo de simulación actual. Estas métricas se utilizarán para generar algunos de los gráficos que mostraremos en la Sección IV. De hecho, también recopilamos estas métricas cuando el primer paquete del flujo se inyecta en la red, y cuando se inyecta en la red el último paquete del flujo. Cuando el último paquete del flujo llega a su destino, el tiempo de simulación actual se compara con las entradas de la tabla hash. De esta manera, generamos un conjunto de métricas útiles, como el tiempo de finalización del flujo (FCT) desde la generación y la inyección del primer y último paquete del flujo para todos los flujos individualmente. Se debe tener en cuenta que estas métricas se pueden usar para generar funciones de distribución acumulativas (CDF) de los tiempos de finalización de los flujos, y también para obtener los valores medios de estas métricas, agregados por una clase de tráfico específica (que es útil para generar gráficos de barras).

El fragmento de código 1 muestra un ejemplo del código XML que debemos incluir en los parámetros para usar el modelo de tráfico propuesto. En este ejemplo, asumimos que la distribución de destinos es común a todas las clases de tráfico, aunque la distribución de destino puede ser diferente para cada clase de tráfico, de acuerdo con los modelos descritos anteriormente. Por ejemplo, la distribución de destinos se puede definir de la siguiente manera en el archivo XML: línea *traffic_class*, en la forma `<traffic_class id = X ... distribution = '' Poisson '' ...>`.

Listing 1: Ejemplo de XML que define el tráfico de una carga de trabajo.

```
<list name="DCApplication">
    <Number_of_classes>4</Number_of_classes>
    <traffic_class id="0" smallest="1" ↩
        biggest="100" percentage="69.52">
    <traffic_class id="1" smallest="100" ↩
        biggest="1000" percentage="25.3">
    <traffic_class id="2" smallest="1000" ↩
        biggest="10000" percentage="3.0">
    <traffic_class id="3" smallest="10000" ↩
        biggest="30000" percentage="2.18">
    <number_flows>10000</number_flows>
    <max_gen_time>2000000.0</max_gen_time> ↩
        <!-- nanoseconds -->
    <destination_dist>Normal</↩
        destination_dist>
</list>
```

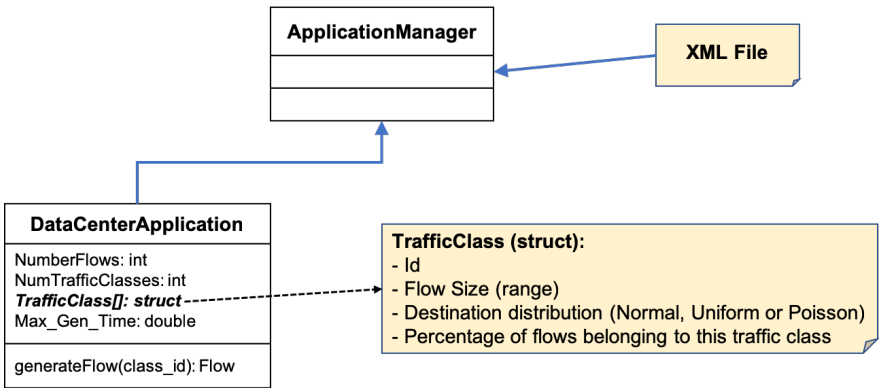


Fig. 1: Diagrama UML de la implementación del modelo de carga de trabajo.

IV. Evaluación y Resultados

En esta sección mostramos resultados de pruebas extensivas para probar y verificar nuestra caracterización de la carga de trabajo. Primero, describimos el modelo de simulación en el cual hemos incluído nuestra implementación, y las configuraciones de red asumidas. Después. mostramos y analizamos los resultados obtenidos de las simulaciones para las configuraciones de red modeladas.

A. Modelo de Simulación

Para los experimentos de evaluación de las cargas de trabajo modeladas, hemos utilizado una herramienta de simulación personalizada basada en eventos escrita en C++. Específicamente, el simulador modela, con precisión a nivel de ciclo (es decir, cada evento requiere X ciclos para finalizar), varios tipos de redes de interconexión al definir su topología, el algoritmo de enrutamiento, las políticas de conmutación y control de flujo, la arquitectura del conmutador y las características de las interfaces de red y de los enlaces. Para todas las configuraciones, asumimos enlaces en serie full-duplex segmentados con 40 Gbps de ancho de banda y 1000 μs de retardo en la propagación del enlace.

Se ha utilizado una configuración de red similar en cada simulación: es decir, una topología de red CLOS con enrutamiento determinístico (*D-mod-K*). Cada conmutador en la red emplea una tecnología de búfer compartido siguiendo el modelo de conmutador de Broadcomm [10], junto con el control de flujo de PFC. Sin embargo, dados los diferentes tamaños de red, los parámetros de los umbrales deben configurarse manualmente para cada tamaño de red. La tabla I especifica las configuraciones para cada tamaño. Se debe tener en cuenta que los valores para el espacio garantizado a la entrada por puerto, por cola, el espacio de reserva a la entrada, el límite de espacio compartido a la salida por puerto, por cola y el espacio garantizado a la salida por puerto y cola son todos iguales para cada tamaño de red, con valores 4096 KB, 4096 KB, 409600 KB, 768000 KB, 384000 KB y 4096 KB, respectivamente. Además, es necesario conocer el significado de estos valores: primero, se llena el espacio garantizado (tanto por puerto como por cola, ya que cada cola está realmente dentro de un puerto). Luego, el espacio compartido comienza a llenarse: todas las colas se asignan al mismo espacio compartido. Finalmente, el espacio de reserva (en caso de que exista) comienza a llenarse.

TABLA I: Configuraciones para el búfer compartido.

#	Network Size	Shared-buffer (SB) Size (KB)	Ingress SB Limit (KB)	Egress SB Limit (KB)
1	1024 nodes	24576	12288	12288
2	4096 nodes	12288	6144	6144

En cuanto a los experimentos, la misma configuración de patrón de tráfico se ha usado para todas las configuraciones de red y funciones de distribución de destinos. Para cada clase de tráfico, hemos definido tuplas $\langle min\_size, max\_size, percentage \rangle$ tal y como sigue:

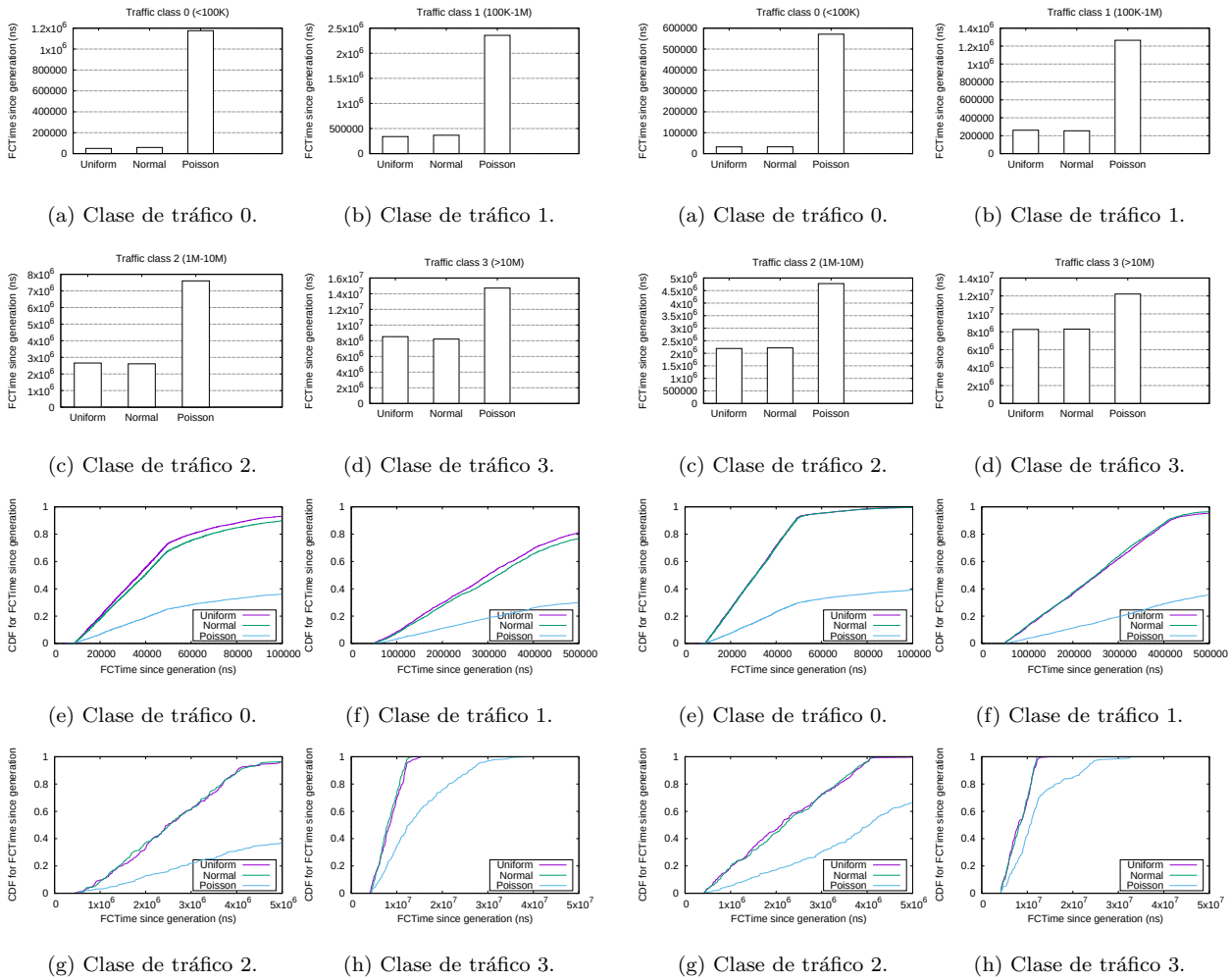
- Clase de tráfico 0: 1KB a 100KB, 69.52% del tráfico total.
- Clase de tráfico 1: 100KB a 1MB, 25.3% del tráfico total.
- Clase de tráfico 2: 1MB a 10MB, 3% del tráfico total.
- Clase de tráfico 3: 10MB a 30MB, 2.18% del tráfico total.

Además, de acuerdo a estos porcentajes, hemos generado 10,000 flujos para cada experimento. El tamaño de MTU (o tamaño de celda) se asume de 1,000 bytes. En cuanto a las funciones de distribución de destinos, hemos usado las distribuciones *Uniforme*, *Normal* y *Poisson*, las cuales han sido explicadas en la Sección II

B. Resultados

La figura 2 muestra los resultados de los FCTs desde generación mediante histogramas y CDFs para la configuración de red #1 (ver Tabla I), para tres funciones de distribución de destinos (es decir, Uniforme, Normal y Poisson).

Como podemos ver, las líneas que representan a las distribuciones Poisson obtienen los peores resultados, ya que generan flujos de tráfico que visitan los destinos más sobre-utilizados. Hay que tener en



(a) Clase de tráfico 0. (b) Clase de tráfico 1.

(c) Clase de tráfico 2. (d) Clase de tráfico 3.

(e) Clase de tráfico 0. (f) Clase de tráfico 1.

(g) Clase de tráfico 2. (h) Clase de tráfico 3.

Fig. 2: Histogramas y gráficos CDF mostrando los FCTs desde generación para las funciones de distribución de destinos Uniforme, Normal y Poisson en una red de interconexión CLOS de 1024 nodos.

(a) Clase de tráfico 0. (b) Clase de tráfico 1.

(c) Clase de tráfico 2. (d) Clase de tráfico 3.

(e) Clase de tráfico 0. (f) Clase de tráfico 1.

(g) Clase de tráfico 2. (h) Clase de tráfico 3.

Fig. 3: Histogramas y gráficos CDF mostrando los FCTs desde generación para las funciones de distribución de destinos Uniforme, Normal y Poisson en una red de interconexión CLOS de 4096 nodos.

cuenta que hay nodos finales de destino que reciben una cantidad de tráfico superior a otros destinos. Esto hace que algunas rutas de la red se obstruyan debido a las rutas sobre-usadas en la red. Por lo tanto, el tráfico que sigue estas rutas obstruirá la red, generando situaciones de incast.

De manera similar, la Figura 3 muestra los resultados del tiempo de FCT desde la generación por medio de histogramas y gráficos CDF para la configuración de red #2 (consultar la Tabla I), para tres funciones de distribución de destinos (las mismas de antes). Como en el escenario anterior, la distribución de Poisson obtiene los peores resultados, ya que termina generando flujos de tráfico que visitan los destinos mas sobre-utilizados de forma recurrente. Por lo tanto, aparecen situaciones de incast que conducen a la congestión y a los escenarios de HoL blocking que degradan el rendimiento de la red.

Nótese que el FCT absoluto en este escenario para la distribución Poisson es inferior al obtenido en la Figura 2. Esto sucede porque la cantidad de flujos de tráfico generados no varía en estas dos configuraciones de red, por lo que la carga de tráfico resultante es mayor en la configuración de red #1 (es decir, 1024 nodos finales) que en la configuración de red #2 (es decir, 4096 final) nodos). Esta observación también se puede confirmar mirando los gráficos CDF. Se debe tener en cuenta que el 90% de los flujos completados finalizan con un FCT más bajo en la Figura 3 que en la Figura 2.

V. Conclusiones

El modelado de las cargas de trabajo en las DCNs es, al mismo tiempo, crucial y difícil. Por un lado, es crucial ya que los diseñadores de redes necesitan entender mejor los cuellos de botella que generan las aplicaciones y los patrones de tráfico de servicio en la DCN. Por otro lado, es difícil, ya que los propietarios de los DCs no siempre publican la información de sus cargas de trabajo, luego es difícil recopilar estos datos para diseñar mejores DCNs. Sobre la base de la información disponible (por ejemplo, de los DCs de Facebook), en este artículo, hemos descrito nuestro modelo para generar cargas de trabajo de tráfico sintético para alimentar nuestros programas de simulación de DCNs. Hemos analizado los

detalles de la información disponible y hemos seleccionado un conjunto de parámetros comunes, como la clase de tráfico (es decir, la aplicación o el servicio), el tamaño de flujo, la distribución de destinos, el tiempo de generación, etc., para caracterizar las cargas de trabajo. Con estos parámetros, hemos diseñado e implementado un generador de flujos de tráfico, que se ha integrado en nuestro programa de simulación. Hemos realizado experimentos de simulación modelando varias configuraciones de DCNs para probar y verificar el modelo. De los resultados obtenidos, podemos concluir que la distribución de destinos es crucial y se requiere conocerla para modelar cargas de trabajo precisas.

Agradecimientos

Este trabajo ha sido apoyado conjuntamente por el Ministerio de Ciencia, Innovación y Universidades y la Comisión Europea (fondos FEDER) bajo el proyecto RTI2018-098156-B-C52 (MCIU/FEDER), por la JCCM bajo el proyecto SBPLY/17/180501/000498, y por la UCLM/FEDER bajo el proyecto 2019-GRIN-27060. Jesús Escudero-Sahuquillo está financiado por la UCLM y la Comisión Europea (fondos FSE), con un contrato para acceder al Sistema Español de Ciencia, Tecnología e Innovación, para la implementación del programa de investigación de la UCLM (fecha de resolución de la UCLM: 31/07/2014).

Referencias

[1] K. Bilial et al., "Quantitative comparisons of the state-of-the-art data center architectures," *Conc. and Comp.: Practice & Experience*, vol. 25, no. 12, pp. 1771–1783, Aug. 2013.

[2] D. Abts and J. Kim, *High performance datacenter networks: architectures, algorithms, and opportunities*, Morgan & Claypool, San Rafael, Calif., 2011.

[3] A. Singh et al., "Jupiter rising: A decade of clos topologies and centralized control in google's datacenter network," in *Proceedings of the 2015 ACM SIGCOMM '15*, London, United Kingdom, 2015, ACM, pp. 183–197.

[4] Paul Congdon, "Ieee 802 nendica report: The lossless network for data centers," Tech. Rep., IEEE-SA Industry Connections, 2018.

[5] Arjun Roy, Hongyi Zeng, Jasmeet Bagga, George Porter, and Alex C. Snoeren, "Inside the social network's (datacenter) network," Tech. Rep., Department of Computer Science and Engineering, University of California, San Diego, 2015.

[6] Theophilus Benson, Aditya Akella, and David A. Maltz, "Network traffic characteristics of data centers in the wild," in *proceedings of the 10th ACM SIGCOMM conference on Internet measurement (IMC '10)*, New York, NY, USA, 2010, pp. 267–280.

[7] S. Chen, S. GalOn, C. Delimitrou, S. Manne, and J. F. MartÃnez, "Workload characterization of interactive cloud services on big and small server platforms," in *2017 IEEE International Symposium on Workload Characterization (IISWC)*, Oct 2017, pp. 125–134.

[8] Yibo Zhu, Haggai Eran, Daniel Firestone, Chuanxiong Guo, Marina Lipshteyn, Yehonatan Liron, Jitendra Padhye, Shachar Raindel, Mohamad Haj Yahia, and Ming Zhang, "Congestion control for large-scale rdma deployments," in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, New York, NY, USA, 2015, SIGCOMM '15, pp. 523–536, ACM.

[9] Xiaofeng Gao, Linghe Kong, Weichen Li, Wanchao Liang, Yuxiang Chen, and Guihai Chen, "Traffic load balancing schemes for devolvedcontrollers in mega data centers," Tech. Rep., Department of Computer Science and Engineering, Shanghai Jiao Tong University, 2016.

[10] "Ns3 simulator for rdma over converged ethernet v2 (rocev2), including the implementation of dcqcn, timely, pfc, ecn and shared buffer switch," Tech. Rep.

Enhanced User Association in Software-Defined WLANs for AP and Channel Load Balancing

Blas Gómez [1], Estefanía Coronado [2], José Villalón[1], Roberto Riggio[2], Antonio Garrido[1]

Abstract— **In traditional Wi-Fi networks, user-AP association is usually driven by the highest signal strength. However, especially in case of very dense deployments, this may lead to uneven user distributions and thus, to poor network performances. Most of the current research addresses this problem by focusing either on (i) maximizing average signal quality; (ii) AP load or (iii) maximizing average throughput. Nevertheless, proposed solutions are hard to implement while preserving compatibility with IEEE 802.11 standard. In this regard, Software Defined Networking (SDN) has recently emerged as a novel approach for network control and management. In this paper, we present an SDN-based solution for joint user association and channel assignment in Wi-Fi networks. It jointly considers the aforementioned indicators as input to make more accurate user association decisions towards higher performance and network efficiency.**

Keywords— **Wi-Fi, SDN, IEEE 802.11, WLANs, mobility management, load balancing**

I. Introduction

Nowadays, modern society is tending to a wireless world. This is in line with the recent trends in the market. As a matter of fact, smartphones, laptops, headphones, smartwatches and many other portable devices are able to use at least one Radio Access Technology (RAT). Such is the concern, that the past years have witnessed a sustained increase in mobile traffic demands that is predicted to reach 77 exabytes per month by 2022 [1]. In this scenario, Wi-Fi has emerged as an efficient way to connect users to the internet due to its low deployment and operational costs. Nevertheless, its contention-based scheme and its unplanned nature lead to suboptimal performances in case of very dense deployments. Moreover, Wi-Fi operates on unlicensed frequency bands. This contributes to its low cost and its ease of deployment but it makes it more vulnerable to interference from co-located deployments as the number of available channels in both the 2.4 GHz and the 5 GHz bands is limited. This produces a throughput degradation when multiple Access Points (APs) are in the same collision domain. This problem aggravates in dense deployments where the number of APs per unit of area increases, thus leading to overloaded channels since frame transmissions must contend for a slot of time.

[1]Instituto de Investigación en Informática de Albacete (I3A), Universidad de Castilla-La Mancha, Albacete (España), e-mail: {Blas.Gomez, JoseMiguel.Villalon, Antonio.Garrido}@uclm.es

[2]Wireless and Networked Systems, FBK CREATE-NET, Trento (Italy), e-mail: {e.coronado, rriggio}@fbk.eu

The IEEE 802.11 standard [2] does not state any algorithm that the stations should use for AP association. This implementation is left to the vendor choice. In general, a client-driven approach is followed, in which stations select the AP with the highest Received Signal Strength Indicator (RSSI). Nonetheless, notice that this factor does not provide information about noise and interference in the wireless medium, and does not consider the resource status. Thus it may lead to uneven load distribution throughout the network which leads to suboptimal network resources allocation. This problem is even worse if we think of heavy users concentrations due to flash events such as an important exam in a university or an autograph session in a mall, where all stations would try to connect to a small set of APs leaving the rest idle.

Academia has proposed different solutions to tackle the aforementioned problems. The majority of them are focused on three different targets: (i) Maximize average signal quality; (ii) Balance traffic loads across the network; (iii) Maximize aggregated network throughput. Regardless of the target pursued, the innovation of the solutions is limited when aiming to preserve the compatibility with the standard. One of the main characteristics of 802.11 is the backward compatibility between the various amendments. Although it has facilitated the coexistence in the market of devices from different generations, it is true that traditional network architectures find difficult to provide improved mechanisms without introducing modifications in the standard. However, a new architecture has recently emerged as a way to redesign network functions: Software Defined Networking (SDN). It decouples data-plane from control-plane by providing high-level programming abstractions, allowing to implement traditional network tasks on top of a centralized controller that has information of the whole network. This improves management and makes possible to dynamically and programmatically adapt the network by centralizing network intelligence. SDN is a consolidated technology in wired networks but solutions intended for wireless networks such as [3] have just started to appear.

In this work, we present a new user association scheme that jointly pursues the three aforementioned targets. In particular, the presented algorithm aims to balance load among all available channels and all available APs while maximizing average

signal strength through effective user reassociation and channel assigning methods.

The rest of the paper is organized as follows. Section II provides an overview of the related work. In Section III we explain the motivation behind the desgin of the approach. In Section IV we describe implementation details of the algorithms. Finally, conclusions are given in Section V.

II. Related Work

Association schemes in WLAN networks are a hot topic in academia, demonstrated by the existing body of literature aiming diverse goals. For instance, authors in [4] try to balance the load among the APs to achieve user bandwidth fairness. For that purpose, each station must specify its bandwidth requirements for the current session. Based on this, stations are associated with the APs that can better fulfil such requirements while maximizing overall traffic. The same target is chased in [5] where authors propose an algorithm that aims to minimize the number of stations per AP based on signal strength. However, the proposal in [4] is set to be more efficient as the number of attached clients is not an accurate estimator of the workload of an AP since each client may generate different amounts of traffic.

Authors in [6] also try to balance load at AP level. By means of an SDN centralized controller, they fill a scoring matrix that takes into account both RSSI and occupancy rate. stations join the APs with the highest score. These approaches include signal strength as part of their decision making. Nonetheless, this may lead to ping-pong effects which are difficult to handle without coordination between the APs. This effect is tackled in [7], where stations periodically look for the most suitable AP in terms of traffic load and signal strength although a handover is not carried out until the same AP has been defined as the best choice for n consecutive times.

Authors in [8] and [9] also propose approaches based on signal strength, but in this case the decission is taken at the station side. This is not optimal as they do not consider network-wide performance. In [10] authors propose a distributed scheme that aims to adapt transmission power of the APs to tune the cell size according to their load and their neighbours' load to balance load producing the migration of stations from the most loaded APs to their less loaded neighbours. This method is known as Cell Breathing. The problem of their proposal resides on the difficulty of the implementation as the APs are in charge of computing their load and communicating it to the rest of APs in the network. A centralized approach using SDN would solve problems in the implementation, making it more efficient. This is the case of [11] where a combination of Cell Breathing and SDN is presented.

Other approaches aim to improve network performance by using network load instead of signal strength. This is the case of [12] where using an SDN architecture they force handovers from most loaded APs to the least loaded ones. It is also the case of [13] where the authors use the average workload of the network to redistribute traffic when a new station joins the network. However, they also look into signal strength in case it deteriorates excessively. Nonetheless, modifications in the standard beacon frames are required. In [14] the AP sends load status reports to a SDN central controller whenever status in the AP changes, i.e., a new station connects to the AP or the difference between the previous and the new load exceeds a threshold. SDN controller computes fairness and if it is not close to 1 the SDN dispatches stations from overloaded APs to underloaded ones.

Handovers usually produce a reassociation process that involves a period of time where the station has no connection. To deal with this problem, authors in [15] propose an approach where association decisions are driven by RSSI of the clients although it requires that APs operate on non-overlapping channels. In [16] authors introduce the concept of Virtual Access Points to allow seamless handovers allowing to set different channels for each AP. In [17] SDN is leveraged for seamless handover together with a centralized reassociation algorithm that is network-wide aware to reduce interpacket delays and provide QoS based on a Markovian analytical model.

Most of the problems we tackle in this paper are aggravated in dense networks such as universities or office buildings where Enterprise WLANs (EWLANs) are usually deployed. In this regard in [18] a solution focused on EWLANs is presented. This SDN-based approach takes into account the number of users per AP and the channel load at AP level. This is inefficient as other APs in the same collision domain also use channel time and could overload the channel, resulting in throughput degradation. Even more metrics are combined in [19] where RSSI, potential capacity, achievable data rate and location of users are considered for association decissions.

Link quality and interference are also very important to unlock the full potential of wireless networks. Usually, stations tend to statically connect using a static configuration even if many of them are available. In [20], authors present a load balancing algorithm that selects different network interfaces for each flow based on required QoS, available bandwidth and link quality to find the best possible path for all flows, maximizing overall throughput in the network. In [21] they also try to maximize throughput in the network. To do that stations seek for the AP with more available bandwidth. This approach seems to be selfish and not optimal. However, seeking for the AP with more available bandwidth implicitly implements least-load-first AP selection.

The number of works on this topic is considerably large, but as shown in this section we can sort them out attending to the main goal they chase to improve network performance. Most of them have one of the following goals: (i) Maximize average signal quality; (ii) Balance traffic loads across the network;

(iii) Maximize aggregated network throughput. Our approach aims to improve network performance by performing a trade-off among the aforementioned goals to get a more versatile approach that performs better in more situations. Furthermore, we also aim to ensure seamless handovers to contribute to higher performances and to minimize interference by means of an efficient channel distribution.

III. Enhanced User Association for AP and Channel Load Balancing

A. Motivation

When two or more APs are in the same collision domain they produce interference and collisions when trying to use the medium at the same time. We say that two APs are in the same collision domain if they are within the carrier sensing range and use the same channel. Interference and collisions are the most important cause of performance degradation in wireless networks. This is deeply discussed in literature as for instance in [22], where authors have shown an strong relationship between channel utilization and network performance. They show how delivery ratio drops when channel occupancy is higher than 60% due to the number of collisions between frames and the decrease in the Modulation an Coding Schemes (MCS) chosen by rate selection algorithms upon several failed transmissions. A lower data rate determines the time each frame keeps the channel busy, i.e., it prevents other frames to be sent over the same channel during more time, reducing throughput.

IEEE 802.11 standard uses Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA) to avoid collisions. In CSMA/CA frames are separated by a gap of time called Inter-Frame Spacing (IFS). This gap is used for carrier sensing before starting transmission. In this regard, the standard defines Distributed Coordination Function (DCF) that was later improved in the IEEE 802.11e with the definition of Hybrid Coordination Function (HCF). Both distributed methods require stations to perform carrier sensing before starting a transmission. In the case of DCF, if the medium is idle for a longer period of time than the duration of a Distributed Coordination Function Inter-Frame Spacing (DIFS), the station is allowed to transmit. Otherwise, the station waits until it gets idle. Then, it again waits a DIFS and a random backoff time. If during backoff time another station starts transmitting, the backoff counter stops. In the next try, the process is repeated but instead of selecting a new backoff time the remaining one is used to avoid those starvation situations. Priority is given to control traffic by defining a shorter IFS called Short Inter-Frame Spacing (SIFS).

In the case of HCF, a new mechanism called Extended Distributed Channel Access (EDCA) is introduced defining 8 Traffic Classes (TCs). Traffic is assigned to a TC attending to QoS reasons. Each TC has a different IFS, known as Arbitration IFS (AIFS). Thus, in both cases, frames transmitted over the same channel either collide or one of them is delayed. In either case, a reduction in the aggregated network throughput is expected. Hence, channel assignment must minimize interference between APs, as well as their number on the same collision domain when possible.

The number of available channels and the number of APs in the same collision domain will determine the efficiency of a channel assignment procedure. The higher the number of available channels, the lower the probability of finding two APs using the same one. Thus, identifying the channels used by the APs in the neighbouring networks is very important. However, the number of available channels may be very limited, especially in congested areas. In the same way, when many clients are connected to the same AP they will have to share the same resources so collisions and frame delays are more likely to happen. Hence, an efficient network resource allocation in terms of both channel assignment and user association is tremendously important.

B. Channel Assignment Algorithm

In light of this, we need a channel assignment algorithm that efficiently assigns resources. For that purpose, we take as a basis the algorithm presented in [22]. The work presents a constraint programming algorithm that recursively tries to assign a channel to the set of APs with the lowest number of available channels, i.e., channels that have not been assigned yet to neighbour APs and do not overlap with the ones already assigned to them. If any channel is available, the algorithm will choose the least used one to reduce interference. This algorithm as it is will be run once at the very beginning to get the more efficient assignment in the network.

The problem of this algorithm is that once channels are assigned they are not adapted to the changing network conditions. Thus, a modified version of the algorithm is used, which is shown in Algorithm 1. It is run when the distribution of channel occupancy is not even and the problem cannot be solved by reassigning clients. In this case, the algorithm will work in the same way but when a channel load is over the threshold, it will be removed from the list of possible channels. A channel will be over the threshold if it is the most loaded one and the condition (Max(`ChLoads`) - Min(`ChLoads`)) > Med(`ChLoads`) is true. This is necessary as only reassigning channels with the same algorithm would just find another distribution that is optimal with regard to interference and minimizing collision domains but occupancy would still be likely uneven as clients would follow the same distribution regarding channels.

C. User Association Algorithm

The main improvement of our approach comes with the user association algorithm shown in Algorithm 2. The SDN-controller gathers uplink RSSI and AP load, measured in channel usage carried out by that AP in bytes. Furthermore, it computes chan-

Algorithm 1 Channel assignment algorithm

Input:

neighbours: graph storing the neighbours of each AP.

channels: list of available channels.

overlaps: dictionary storing the overlapping channels.

Output:

assignment: dictionary of (AP,channel) assignment.

```
1:  procedure SOLVE
2:   remainingAPs ← APs not in assignment
3:   if LEN(remainingAPs)==0 then
4:    return assignment
5:   Sort remainingAPs by the lowest number of
     available channels and the highest number of
     neighbors in assignment
6:   nextAP ← remainingAPs[0]
7:   possibleCh ← channels
8:   for each AP in neighbours[nextAP] do
9:    APCh ← assignment[AP]
10:   possibleCh←possibleCh-APCh
11:   possibleCh←possibleCh-overlaps[APCh]
12:  if not possibleCh then
13:   possibleCh ← MIN(assignment)
14:  for each channel in possibleCh do
15:   assignment[nextAP] ← channel
16:   if CHANNELLOADTHRESHOLD() then
17:    possibleCh ← possibleCh - channel
18:  return SOLVE(neighbours, channel,
     assignment)
```

nel load, i.e., the addition of the channel usage of all APs using the same channel. The algorithm is executed for a specific AP whenever the controller gets an update of the previous statistics from this AP. This period is configurable at the controller side. Average RSSI of an AP refers to the average of the uplink RSSIs of all the stations connected to that AP. The controller stores the values on a list for each one of the aforementioned metrics. Thus, for each one of the metrics, there is a list containing the values of each AP. However, the values in the lists are not the immediate values but the average of the last 10 values reported by the APs. This is done to avoid ping-pong effects, as the situations of uneven load distribution should keep constant over time to have an impact on the average that is big enough to trigger the threshold.

Network is considered as unbalanced whenever the list of average RSSIs, or the list of AP loads fulfil the condition (MAX - MIN) > MED. If this is the case, the SDN-Controller will seek for AP-station associations that can improve the performance. For that purpose, it checks for each client in the network which AP maximizes the value of $\texttt{RSSI}_j \times (\texttt{ChLoad}_{AP_i} + \texttt{LoadAP}_i)$. Bear in mind that RSSI always has a negative value. The larger the number the better the signal quality is. Thus, when maximizing the product the algorithm selects the best trade-off between signal quality and network resources used. A good average signal quality helps to select a better MCS which results in better throughput and thus, the channel keeps busy a shorter time.

To improve load balance two factors are taken into account: the load of the channel that the AP is using and the AP load itself. The first one aims to avoid overloaded channels as clients connected to those channels would experience a significant decrement in performance. The second one is used to avoid overloaded APs. Each AP can serve a certain data rate. When this data rate is full, packets need to be delayed, which results in a poorer performance for stations connected to this AP. Minimizing those effects by moving workload to less congested AP results on a better aggregated throughput. When looking back at Algorithm 2, bigger channel and AP load values produce a smaller value of the product with RSSI so, maximizing it results in the election of the combinations of channel and AP that are less loaded. If there is no better option, the algorithm selects the same AP and no handover will take place. After a handover, the controller keeps track of the performance of the new network distribution. If after a configurable period of time the performance is worse than before, the handover is reverted and the algorithm continues its execution.

Algorithm 2 Load balancing algorithm

Input:

RSSIs: list of average RSSI of each AP.

APLds: list with the loads of each AP.

ChLds: list with the load of each channel.

Output:

NewAP: The AP to migrate to if any.

```
1:  if Max(APLds)-Min(APLds)>Med(APLds) then
2:   for each i in APs do
3:    for each j in ClientsAP_i do
4:     newAP ← Max_j(RSSI_j× (ChLd_APi + LdAP_i))
5:  else if Max(RSSIs)-Min(RSSIs)>Med(RSSIs)
    then
6:   for each i in APs do
7:    for each j in ClientsAP_i do
8:     newAP ← Max_j(RSSI_j× (ChLd_APi + LdAP_i))
9:  else if Max(ChLds)-Min(ChLds)>Med(ChLds)
    then
10:  Reassign channels
```

IV. Implementation Details

A. Overview

The proposed scheme has been implemented taking as a reference the 5G-EmPOWER platform [3]. 5G-EmPOWER is a Multi-access Edge Computing Operating System (MEC-OS) which converges SDN and NFV into a single platform supporting lightweight virtualization and heterogeneous radio access technologies. A high level,

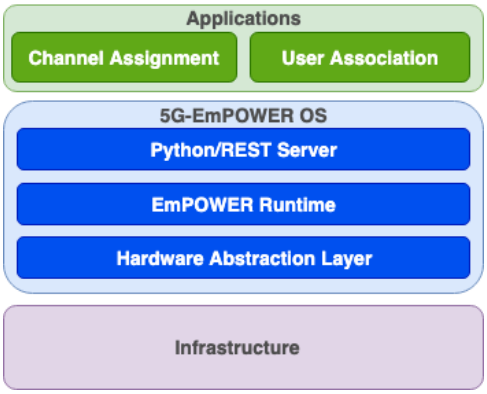


Fig. 1. The 5G-EmPOWER MEC-OS System Architecture

view of the 5G-EmPOWER MEC-OS architecture is sketched in Figure 1.

The 5G-EmPOWER platform consists of a hardware abstraction layer converging different control and management protocols as well as several radio access networks into a set of abstractions that are then exposed to the application layer. In this section, we provide an overview of the Light Virtual Access Point (LVAP) abstraction and the network metrics that are used to implement the algorithm.

B. Seamless Handover Across Different Channels

In [23] the LVAP abstraction is introduced. It provides state management of wireless stations through a high-level interface. A client joining the network triggers the creation of an LVAP that is individual and unique for that station. Hence, each AP handles as many LVAPs as stations that are connected to it. The LVAP handles association, authentication, handover and resource management. This concept allows seamless handovers between APs that operate on the same channel.

Removing an LVAP from an AP and instantiating it on another AP results in a handover. However, this does not work when the APs are tunned on different channels. To avoid this problem, authors in [22] use the Channel Switch Announcement (CSA) defined in IEEE 802.11 standard. CSA announces to stations by the beacon frames that the AP they are connected to is about to switch the channel as well as the new channel. After a configurable number of sent beacons, the channel is switched.

In traditional networks, beacons are sent as broadcast control frames. However, an LVAP sends its own beacons in unicast as it is created for an individual station. Thanks to this, CSAs can be targeted to specific stations through its LVAP. Thus, if a handover to an AP tunned in other channel is needed, the LVAP is instantiated in the target AP and remains inactive until the station connects to the new AP. Source LVAP starts CSA procedure and when this procedure finishes, the source LVAP is removed. At this point, the station would have switched channel and found target LVAP.

C. Network Metrics

Periodically (with a configurable period at the controller side), the APs report the network status to the controller. Such reports contain information about the traffic load, the clients, and the status of the channel among others [24]. In particular, in this paper, we focus on the following metrics:

- RSSI: It is the received signal strength indicator reported, measured at the AP side. Since the proposed algorithm is tailored for uplink traffic, this provides valuable information about the status of the stations.
- Traffic Matrix: It provides the number of bytes and packets transmitted and received by each station.
- Channel Occupancy: It represents the fraction of time the channel is busy at each AP. It is estimated by using rate control statistics and the amount of traffic transmitted and received by the APs.

Other metrics are also provided such as rate control statistics, i.e., MCS, frame delivery ratio in the downlink or estimated throughput in the last observation window. However, they are not used in the proposed algorithm.

V. Conclusion

In this paper, we presented an enhanced user association scheme for Software-Defined WLANs that provides both AP and channel load balancing that addresses the main problems identified in academia to find a trade-off between all of them. The proposal aims to keep a good average RSSI to maintain a high signal quality to avoid the problems derivated from MCSs with low data rates. The network traffic is distributed by combining a channel assignment solution with load balancing techniques at channel and AP levels. Finally, AP served data rate is also limited, so moving stations from the most loaded APs to the least loaded ones is also a very important task of the presented algorithm

Moreover, seamless handover mechanisms for WiFi networks are also addressed as an important research topic, since the time the stations loose connection on a handover impacts on the average network performance. SDN abstractions have been demonstrated as an effective method to perform seamless handovers in multi-channel environments. Moreover, with adequate load balancing algorithms, ping-pong effects can be mitigated.

As future work, we aim to evaluate the performance of the proposed algorithm comparing with traditional RSSI-based association schemes as well as the proposal in [22] as it was the basis for the proposed algorithm.

Acknowledgment

This work has been supported by the Spanish Regional Government of Castilla-La Mancha under the project SBPLY/17/180501/000353 and the Uni-

versity of Castilla-La Mancha under Grant Agreement 2019-GRIN-27060. Moreover, research leading to these results has been supported by the European Union by the EU funded H2020 5G-PPP project under Grant Agreement 761592 (5G-ESSENCE Project).

References

[1] "Cisco visual networking index: Global mobile data traffic forecast update, 2017-2022," Tech. Rep., Cisco, 2019.

[2] "Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications," *ANSI/IEEE Std 802.11 Committee of the IEEE Computer Society Std.*, 2016.

[3] Roberto Riggio, Mahesh K Marina, Julius Schulz-Zander, Slawomir Kuklinski, and Tinku Rasheed, "Programming abstractions for software-defined wireless networks," *IEEE Trans. Net. Service Manag.*, vol. 12, no. 2, pp. 146–162, 2015.

[4] Liang Sun, Lei Wang, Zhenquan Qin, Zhuxiu Yuan, and Yuanfang Chen, "A Novel On-Line Association Algorithm for Supporting Load Balancing in Multiple-AP Wireless LAN," *Mobile Networks and Applications*, pp. 1–12, 2018.

[5] Shiann-Tsong Sheu and Chih-Chiang Wu, "Dynamic load balance algorithm (DLBA) for IEEE 802.11 wireless LAN," *Journal of Science and Engineering*, vol. 2, no. 1, pp. 45–52, 1999.

[6] Fernando Rodrigues de Sa, Adilson Marques da Cunha, and Cecilia de Azevedo Castro Cesar, "Effective AP Association in SDWN Based on Signal Strength and Occupancy Rate," in *Proc. of IEEE 42nd Conference on Local Computer Networks (LCN)*, Singapore, 2017, pp. 159–162.

[7] Li-Hsing Yen and Tse-Tsung Yeh, "SNMP-based approach to load distribution in IEEE 802.11 networks," in *Proc. of IEEE 63rd vehicular technology conference*, Melbourne, Vic., Australia, 2006, pp. 1196–1200.

[8] Murad Abusubaih, James Gross, Sven Wiethoelter, and Adam Wolisz, "On access point selection in IEEE 802.11 wireless local area networks," in *Proc. of 31st IEEE Conference on Local Computer Networks (LCN)*, Tampa, FL, USA, 2006, pp. 879–886.

[9] Nikolaos Papaoulakis, Charalampos Patrikakis, Chryssanthi Stefanoudaki, Platon Sipsas, and Athanasios Voulodimos, "Load balancing through terminal based dynamic AP reselection for QoS in IEEE 802.11 networks," in *Proc. of IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops)*, Seattle, WA, USA, 2011, pp. 600–605.

[10] Eduard Garcia, Rafael Vidal, and Josep Paradells, "Cooperative load balancing in IEEE 802.11 networks with cell breathing," in *Proc. of IEEE Symposium on Computers and Communications*, Marrakech, Morocco, 2008, pp. 1133–1140.

[11] Chia-Ying Lin, Wan-Ping Tsai, Meng-Hsun Tsai, and Yun-Zhan Cai, "Adaptive Load-Balancing Scheme through Wireless SDN-Based Association Control," in *Proc. of IEEE 31st International Conference on Advanced Information Networking and Applications (AINA)*, Taipei, Taiwan, 2017, pp. 546–553.

[12] Keshav Sood, Shigang Liu, Shui Yu, and Yong Xiang, "Dynamic access point association using software defined networking," in *Proc. of International Telecommunication Networks and Applications Conference (ITNAC)*, Sydney, NSW, Australia, 2015, pp. 226–231.

[13] Issam Jabri, Nicolas Krommenacker, Thierry Divoux, and Adel Soudani, "IEEE 802.11 Load balancing: an approach for QoS Enhancement," *International Journal of Wireless Information Networks*, vol. 15, no. 1, pp. 16–30, 2008.

[14] Ying-Dar Lin, Chih Chiang Wang, Yi-Jen Lu, Yuan-Cheng Lai, and Hsi-Chang Yang, "Two-tier dynamic load balancing in SDN-enabled Wi-Fi networks," *Wireless Networks*, vol. 24, no. 8, pp. 2811–2823, 2018.

[15] Yi-Cheng Chan and Dai-Jiong Lin, "The design of an ap-based handoff scheme for IEEE 802.11 WLANs," *International Journal of e-Education, e-Business, e-Management and e-Learning*, vol. 4, no. 1, pp. 72, 2014.

[16] Maria Eugenia Berezin, Franck Rousseau, and Andrzej Duda, "Multichannel virtual access points for seamless handoffs in IEEE 802.11 wireless networks," in *Proc. of IEEE 73rd Vehicular Technology Conference (VTC Spring)*, Yokohama, Japan, 2011, pp. 1–5.

[17] J. Chen, B. Liu, H. Zhou, Q. Yu, L. Gui, and X. (. Shen, "QoS-Driven Efficient Client Association in High-Density Software-Defined WLAN," *IEEE Trans. Veh. Technol.*, vol. 66, no. 8, pp. 7372–7383, 2017.

[18] Yunong Han, Kun Yang, Xiaofeng Lu, and Dongdai Zhou, "An adaptive load balancing application for software-defined enterprise WLANs," in *Proc. of 2016 International Conference on Information and Communication Technology Convergence (ICTC)*, Jeju, South Korea, 2016, pp. 281–286.

[19] Fengming Cao, Zhenzhe Zhong, Zhong Fan, Mahesh Sooriyabandara, Simon Armour, and Ayalvadi Ganesh, "User association for load balancing with uneven user distribution in IEEE 802.11ax networks," in *Proc. of 13th IEEE Annual Consumer Communications & Networking Conference (CCNC)*, Las Vegas, NV, USA, 2016, pp. 487–490.

[20] Tom De Schepper, Steven Latré, and Jeroen Famaey, "A transparent load balancing algorithm for heterogeneous local area networks," in *Proc of. 2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, Lisbon, Portugal, 2017, pp. 160–168.

[21] Li-Hsing Yen, Tse-Tsung Yeh, and Kuang-Hui Chi, "Load balancing in IEEE 802.11 networks," *IEEE Internet Computing*, vol. 13, no. 1, pp. 56–64, 2009.

[22] Estefanía Coronado, Roberto Riggio, José Villalón, and Antonio Garrido, "Wi-Balance: Channel-Aware User Association in Software-Defined Wi-Fi Networks," in *Proc of. IEEE/IFIP Network Operations and Management Symposium*, Taipei, Taiwan, 2018, pp. 1–9.

[23] Lalith Suresh, Julius Schulz-Zander, Ruben Merz, Anja Feldmann, and Teresa Vazao, "Towards programmable enterprise WLANS with Odin," in *Proc. of the first workshop on Hot topics in software defined networks*, Helsinki, Finland, 2012, pp. 115–120.

[24] "The 5G-EmPOWER wiki," [Online]. Available: https://github.com/5g-empower/5g-empower.github.io/wiki, Accessed on 27/05/19.

Control de Congestión Eficiente para Redes HPC con Encaminamiento Adaptativo

Jose Rocher-Gonzalez, Jesus Escudero-Sahuquillo, Pedro J. García y Francisco J. Quiles [1]

Resumen— **La red de interconexión es el elemento principal en los clusters de computación de alto rendimiento (HPC) y centros de datos (DC), donde miles de nodos deben comunicarse de forma rápida y fiable. El rendimiento de la red depende de varias opciones de diseño, como la topología, el algoritmo de encaminamiento, la arquitectura del switch, etc. En la literatura se han propuesto algoritmos de encaminamiento altamente eficientes, ya sean deterministas o adaptativos, para equilibrar de forma inteligente los flujos de tráfico dependiendo de la topología de red, pero su rendimiento se reduce en los escenarios en los que la congestión y sus efectos negativos (por ejemplo, el HoL blocking) aparecen. En particular, en escenarios donde la congestión es intensa y persistente, el HoL blocking puede degradar drásticamente el rendimiento de los algoritmos de encaminamiento adaptativo, ya que pueden extender los flujos de tráfico congestionado por todas las rutas disponibles. Además, como hemos demostrado en estudios anteriores, la dispersión de los flujos congestionados puede deteriorar el rendimiento de los esquemas de colas estáticos utilizados para reducir el HoL blocking mediante la separación de los flujos en diferentes colas del switch buffer. De hecho, como estos sistemas se basan en un criterio estático, definido antes de la inyección del tráfico en la red, no pueden evitar que los flujos congestionados y no congestionados compartan colas cuando se combinan con un encaminamiento adaptativo. En este trabajo, proponemos utilizar algunos esquemas de colas estáticos existentes junto a la asignación dinámica de canales virtuales (VC) para aislar en una solo VC los flujos cuyas rutas han sido encaminadas de forma adaptativa, con el fin de evitar que el impacto de la congestión se extienda a través de varias rutas. Básicamente, los flujos adaptados se mueven a un canal especial de flujos adaptados (AFC), de modo que no interactúan con los flujos asignados a otros VC por el esquema de colas estático. De esta manera, se evita el HoL blocking que los flujos adaptados podrían causar a los flujos no adaptados, incluso si los flujos congestionados se han extendido a través de varias rutas. Por otro lado, el esquema de colas estático reducirá sin ninguna interferencia el HoL blocking que puede aparecer entre los flujos no adaptados. Para evaluar nuestra propuesta hemos realizado experimentos de simulación modelando grandes redes de interconexión basadas en la topología Fat-tree. De los resultados obtenidos, podemos concluir que nuestra técnica reduce de manera eficiente y significativa el impacto del HoL-blocking en las redes de interconexión utilizando encaminamiento adaptativo y esquemas de colas cuando aparece la congestión.**

Palabras clave— **Redes de Interconexión, Control de Congestión, Esquemas de Colas, Encaminamiento Adaptativo**

I. Introducción

En la actualidad, los sistemas de computación de alto rendimiento (HPC) y centros de datos (DC) están compuestos por miles de nodos que trabajan en paralelo para hacer frente a las crecientes demandas, potencia de cálculo y procesamiento de datos, de aplicaciones y servicios utilizados en diversos campos como la genómica, la previsión climática, la inteligencia artificial, la robótica, las redes sociales, etc. En todos estos sistemas, la red de interconexión juega un papel esencial, ya que debe ofrecer un alto ancho de banda y una baja latencia para la comunicación entre los nodos, de lo contrario se convertiría en el cuello de botella del sistema. Para satisfacer las demandas de comunicación, los diseñadores de redes de interconexión se enfrentan a múltiples retos de diseño, como la configuración de topologías de red eficientes y algoritmos de encaminamiento. Concretamente, hay varios requisitos que deben tenerse en cuenta al diseñar la topología de red, como la diversidad de caminos, el bajo diámetro, las condiciones para evitar interbloqueo, la conectividad total, el ancho de banda de la bisección, el coste, etc. En ese sentido, las topologías Fat-Tree [1] se utilizan en sistemas reales de HPC y DC, ya que ofrecen un alto ancho de banda de bisección, diversidad de rutas mínimas y un patrón de conexión que evita los interbloqueos de forma natural. Además, tienen capacidades inherentes de tolerancia a fallos. Gracias a estas características, los Fat-trees se utilizan en grandes supercomputadores como el IBM Summit, el número uno de la última lista TOP500 (noviembre de 2018) desplegando un Fat-tree de tres etapas.

Además, se han propuesto algoritmos de encaminamiento eficientes para los Fat-tree que explotan sus propiedades. Por un lado, los algoritmos deterministas siempre seleccionan la misma ruta entre un par de nodos, pero no tienen en consideración ninguna información sobre el estado de la red o las condiciones de tráfico. Las políticas de encaminamiento deterministas [2], [3] son ampliamente utilizadas porque son fáciles de implementar, y equilibran eficientemente los flujos de tráfico entre las rutas disponibles en la red. Por otro lado, con los algoritmos adaptativos [4], [5], [6] se selecciona la ruta entre dos nodos entre el conjunto de rutas disponibles, y esta selección puede variar dependiendo de las condiciones de tráfico y del estado de la red.

Sin embargo, incluso cuando se utilizan topologías y algoritmos de encaminamiento eficientes, la congestión puede aparecer y degradar severamente el rendimiento de la red. La congestión se produce cuando varios flujos de tráfico solicitan simultáneamente el acceso al mismo recurso de red (por ejemplo, buffers, enlaces, interfaces de red, etc.), ya que se concede a un flujo el acceso al recurso solicitado, mientras que los demás deben esperar. Si esta situación persiste

[1]Departamento de Sistemas Informáticos, Universidad de Castilla-La Mancha, España. e-mail:{jose.rocher, jesus.escudero, pedrojavier.garcia, francisco.quiles}@uclm.es

en el tiempo, entonces la congestión se propaga por toda la red, llegando finalmente a los nodos fuente, debido a la presión hacia atrás que ejerce el control de flujo.

Esta propagación genera estructuras conocidas como árboles de congestión [7] que consisten en flujos de tráfico congestionados conectados. Los principales efectos negativos de la congestión son el *Head-of-Line(HoL) blocking* [8] y el *buffer hogging* [9].

En consecuencia, se han propuesto muchas técnicas para hacer frente al fenómeno de la congestión y sus efectos negativos. Una de las soluciones se basa en la utilización de varias colas en los puertos de los switches para almacenar diferentes flujos por separado, reduciendo así el HoL blocking. Este enfoque puede llamarse separación de flujos basados en colas. Básicamente, los buffers de los puertos de entrada y/o salida se dividen lógicamente en colas que normalmente se implementan como canales virtuales (VCs) [10] (o carriles virtuales (VLs) en la especificación InfiniBand). Cada cola tiene su propio espacio en el buffer y el control de flujo se realiza a nivel de cola.

Estas técnicas, llamadas esquemas de colas estáticos (SQS), son fáciles de implementar y su eficiencia depende de la política de mapeo específica y del número de colas por puerto. Las políticas de mapeo pueden ser "agnósticas" a la topología y encaminamiento, como VOQnet [11], DAMQs [12] o DBBM [13], o basadas en la topología y encaminamiento, como OBQA [14], vFtree [15] y Flow2SL[16], estas últimas utilizan los recursos de la red de forma más eficiente para reducir el HoL blocking.

Las soluciones descritas para la separación de flujos basadas en colas se proponen, en general, para topologías de red que utilizan encaminamiento determinista. Sin embargo, el encaminamiento adaptativo también es ampliamente utilizado. En este sentido, se realizó un estudio preliminar para analizar el impacto del uso del encaminamiento adaptativo sobre la eficiencia de SQSs inicialmente propuestos para escenarios de encaminamiento determinista [17]. En ese estudio, demostramos que, bajo escenarios de congestión intensa, el encaminamiento oblivious y adaptativo puede extender los árboles de congestión a través de las rutas de red disponibles, de modo que los paquetes congestionados pueden terminar siendo almacenados por la mayoría de las colas definidas por el SQS. Por lo tanto, la probabilidad de HoL blocking aumenta y el rendimiento de la red se degrada. Para evitarlo, propusimos restringir la adaptatividad del encaminamiento mediante umbrales de ocupación en las colas para activar la adaptatividad, de modo que se redujera la propagación de la congestión. Los resultados obtenidos mejoran el comportamiento de la red en situaciones de congestión, pero creemos que el rendimiento de la red puede mejorarse aún más.

Para ello, en este trabajo proponemos una nueva técnica que combina las técnicas de separación de flujos basadas en colas con algoritmos de encaminamiento adaptativo, llamada Aislamiento de Flujo Adaptado (AFI). Básicamente, los flujos de tráfico que no han sido encaminados de forma adaptativa se asignan a un conjunto de colas de acuerdo con un SQS. Sin embargo, cuando la ocupación de una cola supera un determinado umbral, los paquetes almacenados en esta cola se encaminan a través de rutas alternativas y, en el siguiente salto, estos paquetes se almacenan en un VC especial, llamado canal de flujo adaptado (AFC), que no está en el conjunto de colas utilizado por el SQS. Desde este salto, los paquetes adaptados permanecen almacenados en los AFCs para evitar que los flujos congestionados en una ruta alternativa compartan colas con los no adaptados. Además, una vez que un flujo se encamina de forma adaptativa y realiza un salto, se encamina siguiendo rutas deterministas, de modo que se restringe la propagación de la congestión. Una vez que la congestión desaparece, el encaminamiento adaptativo deja de enviar paquetes a través de rutas alternativas.

Hemos evaluado AFI en grandes Fat-trees que conectan hasta 11K nodos, a través de experimentos basados en simulación realizados bajo patrones de tráfico sintéticos y basados en trazas. Los resultados obtenidos muestran que AFI reduce significativamente la probabilidad de que los flujos congestionados y no congestionados compartan colas, así como de que los árboles de congestión se extiendan debido al encaminamiento adaptativo, lo que en general reduce significativamente el HoL blocking en comparación con otros enfoques.

El resto de este documento está organizado de la siguiente manera. La sección II resume los antecedentes en cuanto a la topología Fat-tree, los algoritmos de encaminamiento, la congestión y sus soluciones. En la sección III identificamos los problemas que pueden aparecer cuando se combinan esquemas de colas con el encaminamiento adaptativo. La sección IV describe la técnica de aislamiento de flujo adaptado. En la Sección V, explicamos el modelo de simulación y mostramos y analizamos los resultados obtenidos para evaluar nuestra propuesta.

II. Conceptos Básicos

A. Fat-Trees

El Fat-Tree [1] es una topología ampliamente utilizada en los sistemas HPC y DC. Esta familia de topologías multietapa ofrece algunas características deseables y una gran variedad de patrones de conexión. Se han propuesto varios patrones de conexión, como los Real Life Fat-Trees (RLFTs), una subclase de los Parallel Ports Generalized Fat-Trees (PGFTs) [18]. Los RLFTs ofrecen un coste reducido por puerto de switch y, como consecuencia, muchos sistemas comerciales utilizan este patrón de conexión. Hay que tener en cuenta que este patrón de conexión utiliza switches con el mismo número de puertos P. En esta topología, cada switch conecta K ($K = P/2$) puertos con otros switches en la siguiente etapa (K es la aridad del switch, es decir, la mitad del número de puertos). El número de nodos N en un RLFT con n etapas y con una aridad de switch K viene dado

por $N = 2(K^n)$, y el número de switches se puede calcular con la fórmula $S = (N \cdot (2n-1))/2K$.

En general, las topologías RLFT tienen las siguientes propiedades:

- Son libres de bloqueos si se utiliza un encaminamiento de ruta mínima, es decir, todas las rutas utilizadas por el algoritmo de encaminamiento ofrecen el menor número de saltos.
- Ofrecen una amplia diversidad de rutas entre cualquier par de nodos, de modo que hay varias rutas de ruta mínima disponibles entre cualquier par de nodos.
- Ofrecen un ancho de banda de bisección constante (CCB) entre todas las etapas.
- Proporcionan de forma inherente capacidad para tolerancia a fallos, debido a la diversidad de caminos.

El encaminamiento en los RLFTs puede dividirse en dos fases: subida y bajada. Primero, en la fase de subida se pueden seleccionar todos los puertos K de un switch, como puertos de salida, para encaminar un paquete determinado a la siguiente etapa. Los paquetes se dirigen en dirección ascendente hasta que llegan a la etapa de giro. Desde la etapa de giro, sólo hay un camino hacia abajo posible para que ese paquete llegue a su destino. Por lo tanto, el conjunto de rutas de recorrido mínimo está limitado por la aridad del switch K, y el número de etapas n. En consecuencia, la máxima diversidad de rutas de camino mínimo entre cualquier par de nodos es K^{n-1}.

Existen varios algoritmos de encaminamiento propuestos para los Fat-trees, ya sean deterministas o adaptativos. Por un lado, existen algoritmos de encaminamiento determinista que aprovechan la diversidad de rutas, como DESTRO (Deterministic destination and STage based Routing) [2] o D-mod-K [3], que reparte inteligentemente los flujos entre los caminos disponibles de subida.

Sin embargo, el encaminamiento determinista obliga a los paquetes a seguir siempre la misma ruta entre un par de nodos dado. Esto hace que los algoritmos de encaminamiento determinista funcionen eficientemente cuando se generan patrones de tráfico uniformes en la red (es decir, todos los nodos generan una cantidad similar de tráfico que se distribuye entre los nodos de una manera justa). Por otro lado, el encaminado adaptativo es particularmente útil para equilibrar la carga cuando el trafico no es uniforme y existen propuestas como [4], [5], [6] para implementarlo en los Fat-trees.

La Figura 1 muestra las posibles rutas de camino mínimo entre dos nodos en un RLFT de 3 niveles. Las flechas en negrita representan la ruta utilizada por el encaminamiento determinista D-mod-K. Las líneas discontinuas muestran las rutas disponibles utilizadas por los algoritmos de encaminamiento adaptativo, además de la ruta determinista, que también puede ser seleccionada por los algoritmos de encaminamiento adaptativo.

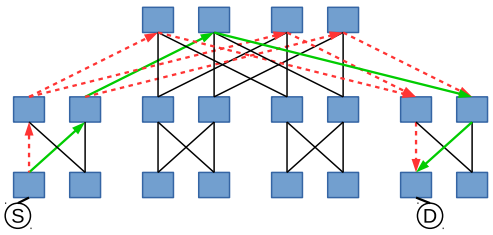


Fig. 1: Rutas posibles entre dos nodos en un RLFT de 3 etapas (k=2). Las líneas verdes representan la ruta D-mod-K. Las líneas rojas representan las rutas del encaminamiento adaptativo.

B. La congestión y sus efectos

La congestión de la red aparece por varias causas, tales como el tráfico Hotspot, ráfagas de tráfico, nodos populares que normalmente reciben mucho tráfico, etc. Independientemente de la causa, la congestión termina ralentizando los flujos de tráfico, reduciendo drásticamente el rendimiento de la red. Específicamente, el origen de la congestión es la contención, que ocurre dentro de un switch cuando varios flujos, desde diferentes puertos de entrada, solicitan de forma simultánea acceso al mismo puerto de salida. Cuando la contención persiste en el tiempo, entonces aparece la congestión, así como sus efectos negativos.

El problema más importante derivado de la congestión es el HoL blocking [8], que en general aparece cuando un paquete congestionado en la cabeza de una cola impide que otros paquetes almacenados detrás avancen, incluso si solicitan puertos disponibles dentro de un switch. Cuando las colas que contienen paquetes congestionados en su cabecera se llenan, la congestión se propaga a otros switches debido a la presión hacia atrás del control de flujo [1] y pueden llegar a cualquier puerto de la red, formando lo que se conoce como un árbol de congestión. Específicamente, la raíz del árbol de congestión se origina en el puerto de salida del switch congestionado y la congestión se propaga desde la raíz hasta las hojas (hasta el NIC del emisor). Apreciamos diferentes tipos de HoL blocking dependiendo del lugar donde se origina la congestión. Nos referimos a low-order HoL blocking [19] cuando los paquetes se bloquean en el mismo switch donde se origina la congestión. Nos referimos a high-order HoL blocking cuando los paquetes se bloquean en un switch diferente del switch donde se origina la congestión, debido a la presión hacia atrás del control de flujo.

Además, la congestión puede producir buffer hogging [9] cuando los flujos congestionados ocupan la mayor parte del espacio de buffer disponible en cualquier puerto de switch, de tal modo que impiden que otros flujos lleguen a ese puerto. También podemos distinguir diferentes tipos de buffer hogging, como el inter buffer hogging o el intra buffer hogging. El in-

[1] Asumimos redes sin pérdidas, donde no se permite descartar paquetes. Estas redes utilizan un control de flujo a nivel de enlace basado en créditos que evita descartar paquetes cuando los buffers que se solicitan están llenos y su posterior reenvío

ter buffer hogging se produce cuando la ocupación de una cola aumenta hasta consumir todo el espacio de buffer compartido entre colas, mientras que el intra buffer hogging se produce cuando un flujo consume todo el espacio asignado a una cola determinada, impidiendo que otros flujos se almacenen en esa cola. En resumen, el HoL blocking y el buffer hogging pueden degradar severamente el rendimiento de la red de interconexión, a menos que se tomen algunas contramedidas.

C. Esquemas de colas

Una de las soluciones tradicionalmente ofrecidas para tratar la congestión es la introducción de innovaciones en la arquitectura del switch para mitigar el HoL blocking y el buffer hogging. Una de estas innovaciones consiste en dividir el espacio del buffer de los puertos de switch en canales virtuales (VCs). Aunque los VCs fueron introducidos originalmente para evitar bloqueos, también pueden ser usados para proporcionar calidad de servicio (QoS) [20] o reducir el HoL blocking [15]. En algunos casos, los buffers de los switches pueden implementar *Virtual Output Queues* (VOQs) [12] a nivel físico, por medio de entradas demultiplexadas en el crossbar interno del switch. De hecho, los switches con VOQs evitan el low-order HoL blocking de forma nativa porque almacenan paquetes dirigidos a diferentes puertos de salida en diferentes VOQs, y los paquetes bloqueados almacenados en una VOQ no retrasan el avance de los paquetes dirigidos a otras VOQs diferentes. Sin embargo, los switches basados en VOQ todavía sufren de high-order HoL blocking. Nótese que los switches basados en VOQ también pueden utilizar esquemas de colas (mediante VCs) de forma ortogonal a los VOQs, con el fin de tratar el HoL blocking de forma más eficiente [21]. También hay que tener en cuenta que el control de flujo en los switches basados en VOQ se realiza a nivel VC, de modo que no es posible controlar la ocupación de una VOQ que está ocupando todo el espacio del buffer, debido a una situación de congestión. Por lo tanto, se necesitan otras estrategias para eliminar el buffer hogging y el HoL blocking.

El uso de los esquemas de colas estáticos (SQSs) ha sido ampliamente estudiado porque son alternativas rentables para reducir el HoL blocking y el buffer hogging. Como se ha mencionado anteriormente, los esquemas de colas almacenan diferentes flujos de paquetes en diferentes colas (o VCs) en los puertos del switch, de acuerdo con una política específica de mapeo de flujo a cola. Esta política de mapeo se define como *estática* ya que se aplica antes de que los paquetes se inyecten en la red, y sin considerar ninguna información sobre el estado de la red. La calidad de la política de mapeo puede ser medida [16], y obviamente determina la eficiencia del esquema de colas. Basándose en esta idea, existen esquemas de colas basados en la topología y el encaminamiento, ya que la calidad de su política de mapeo es alta y coincide con la topología y las propiedades de encaminamiento. Por el contrario, hay esquemas de colas agnósticos a la topología y al encaminamiento que no prestan atención a estas propiedades.

En este trabajo nos centramos en tres SQSs que pueden ser utilizados en Real Life Fat-Trees(RLFTs): *Destination-Based Buffer Management* (DBBM) [13], Flow2SL [16] y vFtree [15]. Concretamente, DBBM es un esquema agnóstico a la topología y al encaminamiento que asigna paquetes con destinos consecutivos a diferentes colas en cualquier puerto de switch, mientras que los paquetes con el mismo destino son almacenados en la misma cola. La política de mapeo de DBBM se define con un función módulo $D \bmod Q$, donde D es el ID destino del paquete y Q es el número de colas (o VCs) por puerto. Como tenemos un número limitado de colas (o VCs), DBBM termina almacenando paquetes para un subconjunto de destinos en cada cola. Téngase en cuenta que la política de asignación de DBBM sólo se basa en el ID destino del paquete para seleccionar una cola.

Por otro lado, tanto vFtree como Flow2SL son esquemas que tienen en cuenta la topología y el encaminamiento, especialmente diseñados para Fat-trees que utilizan algoritmos de encaminamiento deterministas, como "D-mod-K"' [2], [3]. Básicamente, vFtree [15] observa los switches hoja de origen y destino de cada paquete, y baraja los paquetes entre las Q colas disponibles. vFtree almacena en la misma cola los flujos de paquetes que tienen idéntico switch hoja origen y están dirigidos al mismo switch hoja destino, mientras que otros paquetes que tienen el mismo switch hoja origen pero que están dirigidos a un switch hoja destino diferente se barajan entre las Q colas disponibles. Téngase en cuenta que los paquetes dirigidos a switches hoja destino consecutivos se almacenan en colas (o VCs) consecutivas. Desafortunadamente, la política de mapeo de vFtree hace que los paquetes dirigidos a diferentes switches hojas compartan colas en Fat-trees con más de dos etapas. Por lo tanto, para superar este defecto se propuso la técnica Flow2SL [16]. Básicamente, Flow2SL divide la red en grupos de nodos consecutivos y hay tantos grupos como número de colas disponibles (Q). De este modo asigna paquetes a diferentes colas dependiendo de su grupo origen y destino. En otras palabras, los movimientos que tienen lugar entre el mismo grupo fuente y el mismo grupo de destino se asignan a la misma cola (o VC), mientras que los movimientos que tienen entre el mismo grupo origen pero se dirigen a grupos destino diferentes se asignan a colas diferentes. Tenga en cuenta que el HoL blocking entre los flujos asignados a la misma cola se reduce a medida que el número de flujos que comparten colas disminuye a través de los niveles del Fat-tree, debido al balanceo de destinos que ofrece el algoritmo de encaminamiento.

El problema de los esquemas de colas propuestos para reducir el HoL blocking es que no reaccionan bien cuando el algoritmo de encaminamiento es adaptativo, es decir, cuando los flujos pueden ser encaminados a través de todas las rutas disponibles. En este

caso, cuando aparece la congestión, el algoritmo de encaminamiento hace que los paquetes eviten las zonas congestionadas utilizando rutas alternativas, con el fin de evitar estas zonas. El problema es que esta política hace que los árboles de congestión se propaguen también a través de todas las rutas alternativas seleccionadas por el algoritmo de encaminamiento adaptativo. Por lo tanto, cuando se utiliza el encaminamiento adaptativo, las soluciones propuestas para tratar el HoL blocking pueden ser no efectivas, como se describe en la siguiente sección.

III. Planteamiento del Problema

Con el fin de analizar los inconvenientes de las técnicas para tratar los problemas de la congestión propuestas previamente, en un trabajo anterior hemos estudiado la combinación de algoritmos de encaminamientos adaptativos y oblivious con esquemas de colas estáticos en Real Life Fat-Trees (RLFTs) [17]. Básicamente, analizamos la efectividad de los esquemas de colas descritos en la Sección II-C cuando se utilizan algoritmos encaminamientos adaptativos o oblivious. De hecho, observamos que aplicando adaptatividad a los árboles de congestión se generan flujos de tráfico congestionados en toda la red, creando nuevas situaciones de congestión en diferentes puntos de la red. Figura 2 muestra un árbol de congestión generado cuando varios nodos generan un patrón de tráfico "many-to-one", dirigido a un destino "HS".

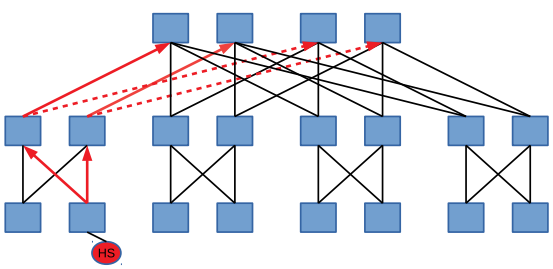


Fig. 2: Situación de congestión en un RLFT de 3 niveles($k = 2$) usando encaminamiento adaptativo. Las flechas rojas muestran el crecimiento de los árboles de congestión que aparecen debido al encaminamiento adaptativo.

En esta situación, los paquetes dirigidos al destino "HS" pueden seleccionar rutas alternativas en dirección ascendente, para evitar los switches en la tercera etapa afectados por los árboles de congestión (flechas rojas en negrita). Como suponemos un patrón de tráfico "muchos a uno", la raíz de la congestión no se mueve a otros puntos finales, y los paquetes dirigidos a "HS" pueden seleccionar rutas alternativas, contribuyendo a generar nuevas ramas del árbol de congestión (flechas rojas discontinuas). Tenga en cuenta que el encaminamiento adaptativo esparce la congestión cuando la congestión es generada por nodos populares que reciben tráfico de muchos nodos fuente. Además, los árboles de congestión no se eliminan, sino que se hacen más fuertes y están más presentes en las rutas de la red, incrementando el HoL blocking y el buffer hogging, degradando aún más el rendimiento de la red. En general, cuando los patrones de tráfico "many-to-one" generan la congestión, los flujos de tráfico adaptados pueden converger y generar situaciones de congestión más fuertes afectando a otras partes de la red.

Por otro lado, la combinación de los algoritmos de encaminamiento adaptativos con los esquemas de colas estáticos (SQSs) implica que se mapearán más destinos por cada VC, ya que este encaminamiento puede seleccionar más rutas que el determinista y por lo tanto habrán más destinos que compartan enlaces. Este incremento de flujos por enlace incrementa la posibilidad de que se genere HoL blocking en los buffers del switch. Además, los SQSs pierden su efectividad para separar flujos en colas y así evitar el HoL blocking y el buffer hogging. Por lo tanto, con el objetivo de ofrecer un algoritmo adaptativo con los beneficios de los SQSs, necesitamos evitar que los algoritmos de encaminamiento esparzan los arboles de congestión por todos los VCs disponibles.

IV. Aislamiento del Flujo Adaptado

En esta sección, proponemos una nueva técnica llamada Aislamiento del Flujo Adaptado (AFI) que contrarresta las desventajas que generan los algoritmos adaptativos esparciendo la congestión por la red. Pensamos que al limitar la adaptatividad de los paquetes congestionados podemos reducir la congestión que se esparce por rutas alternativas. Básicamente, queremos reducir, tanto como sea posible, las situaciones donde la política de encaminamiento adapte los paquetes para evitar los puntos congestionados en la red. Concretamente, en los buffers del switch AFI, combinamos un esquema de colas estático (SQS) con un VC especial, llamado AFC , el cual se usa para almacenar los paquetes que contribuyen a la congestión. AFI detecta la congestión cuando el algoritmo de encaminamiento adaptativo selecciona una ruta alternativa y marca ese paquete como "adaptado". Aquellos paquetes adaptados son movidos al AFC en el siguiente switch y desde ese momento seleccionan rutas deterministas con el objetivo de que el encaminamiento adaptativo no esparza más este flujo congestionado. Sin embargo, los paquetes "no adaptados" son almacenados en otros VCs siguiendo la política de mapeo especifica del SQSs y son encaminados de forma determinista. Por lo tanto, AFI ajusta de forma dinámica rutas alternativas solo una vez por paquete, si la ruta del paquete atraviesa un área congestionada. En las siguientes secciones describimos los requerimientos de la arquitectura de switch que necesita AFI, como opera y sus detalles de implementación.

A. Arquitectura del Switch

Para simplificar la descripción de la técnica AFI, de ahora en adelante, asumimos el uso de Input Queued (IQ) switches, donde los buffers se colocan en los puertos de entrada. Sin embargo, el uso de AFI no solo está restringido a los IQ switches, sino que

también funciona en otras arquitecturas de switch. La Figura 3 muestra la arquitectura de conmutador asumida.

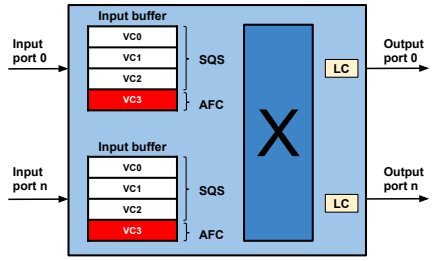


Fig. 3: Arquitectura del switch propuesta para AFI.

Hay al menos dos VCs en el buffer para almacenar paquetes no adaptados, y un AFC por buffer para almacenar los paquetes adaptados. También asumimos que los switches utilizan una política de control de flujo basada en créditos a nivel de VC. Por lo tanto, la lógica del Link Controller (LC) es consciente de los créditos disponibles en el buffer del puerto siguiente. Si la ocupación de un determinado VC a seleccionar por el encaminamiento determinista supera un determinado umbral, esta información es conocida inmediatamente por el LC, ya que realiza un seguimiento de los créditos disponibles por VC. Basándose en esta información, se selecciona un puerto de salida alternativo si el seleccionado de forma determinista tiene el VC congestionado.

Por otro lado, si el algoritmo de encaminamiento no necesita de una ruta alternativa por que los VCs no están lo suficientemente llenos, los paquetes no adaptados son almacenados en los VCs de forma acorde a la política de mapeo de un SQS. De esta forma, nuestra técnica puede ser combinada con los SQS propuestos previamente y es compatible con sus políticas de mapeo especificas siempre que la tecnología de red permita asignar paquetes con algún Service Level o identificador de clase de tráfico [2] Por lo tanto, mientras la congestión no este presente en la red o lo haga de una forma moderada, los SQS son los encargados de separar los flujos de tráfico en VCs. Cuando la congestión se hace persistente y se propaga a otros switches, AFI almacena los paquetes de flujos adaptados en el AFC de los buffers que harán uso de este VC hasta que lleguen al destino. En este caso, los paquetes almacenados en el AFC son encaminados utilizando un algoritmo determinista con el objetivo de no adaptar paquetes que puedan contribuir potencialmente a la congestión. Por otro lado, cuando la ocupación del siguiente buffer de la ruta adaptada baja por debajo del umbral, se deja de adaptar la ruta y se vuelve a utilizar la política de los SQS almacenando los paquetes en los VCs asignados.

Además, podemos asumir que los switches implementan Virtual Output Queues (VOQs), descritos en II-C. En ese caso, los paquetes encaminados hacia el mismo puerto de salida serán almacenados en una VOQ física, evitando así el low-order HoL-blocking. Nótese que cada cada uno de estos paquetes puede estar almacenado en un VC diferente, dependiendo de la política de mapeo del SQS. Por lo tanto, una rama del árbol de congestión puede crecer dentro de una VOQ y, ortogonalmente, a todos los VCs del mismo buffer ya que el control de flujo se realiza a nivel de VC y no a nivel de VOQ. En el caso de los switches basados en la arquitectura VOQ, el buffer hogging puede degradar mucho el rendimiento, ya que todos los VCs del mismo puerto pueden estar llenos de paquetes congestionados. AFI también ayuda a aliviar estas situaciones debido a que condiciona la selección del puerto de salida según un umbral de ocupación del VC en el buffer siguiente, adaptando los paquetes que contribuían a la congestión y almacenándolos solo en el AFC.

B. Descripción de Funcionamiento AFI

La figura 4 muestra como opera AFI cuando aparece la congestión. En esta figura se puede ver una muestra de la red con 3 Switches (A, B y C). En el switch B, la ocupación del VC0 en el puerto de entrada 0 alcanza el umbral de congestión y, desde ese momento, se considera un VC congestionado (Evento #1). Cuando esto sucede, como hemos asumido una política de control de flujo a nivel de VC, el controlador del enlace (LC) del puerto de salida 0 del switch A es consciente de la situación de congestión (Evento #2). Desde ese momento, en el switch A todos los paquetes del VC0 que soliciten ir al switch B por el puerto de salida 0, serán encaminados por un puerto alternativo. Esto sucede cuando un paquete "X" necesita ser encaminado por el puerto de salida 0 del switch A (Evento #3). En ese momento, la lógica de encaminamiento consulta al LC sobre la ocupación del puerto de entrada 0 del switch B. Como el VC está congestionado, en el switch A la lógica de encaminamiento selecciona una ruta alternativa para el paquete "X", encaminándolo por el puerto de salida 1. En este ejemplo, asumimos que se puede encaminar por el puerto de salida 1 del switch A (Evento #4), pero el algoritmo de encaminamiento debe buscar por todos los puertos de salida y seleccionar el más deseable (menor tasa de ocupación del AFC). Una vez el paquete es encaminado sobre la ruta alternativa, el paquete es marcado como "adaptado". Esto se puede implementar mediante la reserva de 1 bit en la cabecera del paquete. De esta forma, cuando llegue al switch C será almacenado en el AFC en el puerto de entrada 0 (Evento #5). Como hemos descrito previamente, una vez el paquete "X" ha sido adaptado, este seguirá una ruta determinista y siempre será almacenado en el AFC.

Por lo tanto, AFI previene el esparcimiento de los flujos adaptados por la congestión sobre rutas alternativas, ya que una vez se detecta la congestión en la red este limita la adaptatividad. Con AFI los flujos de tráfico "adaptados" son encaminados luego de

[2]La terminología Service Level (SL) es empleada por la especificación Infiniband para referirse a las clases o prioridades del tráfico. Otras tecnologías de red, como Intel Omni-Path, usan la terminología Traffic Classes (TC) para referirse a este identificador.

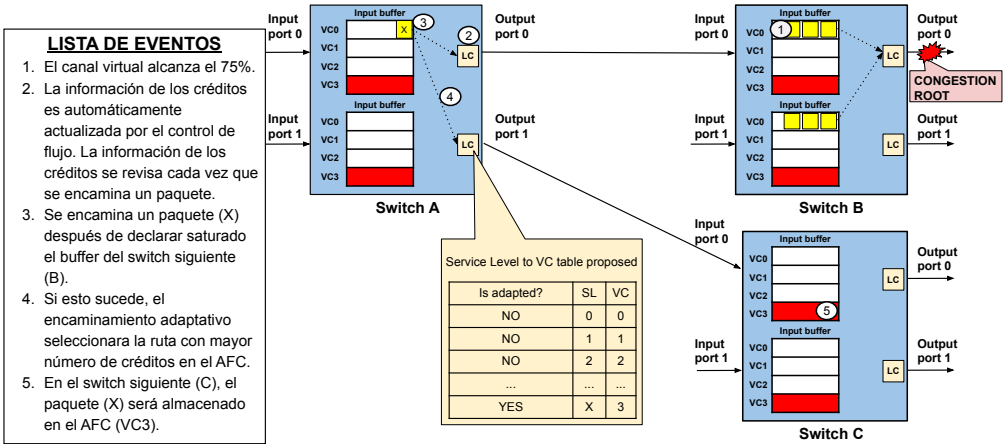


Fig. 4: Ejemplo de funcionamiento de AFI.

forma determinista, mientras que los flujos de tráfico "no adaptados" son almacenados en los VCs de forma acorde a la política del SQS.

C. Detalles de Implementación

AFI puede ser implementado en tecnologías de red comerciales actuales con un mínimo rediseño y sin añadir una gran complejidad a la arquitectura de los switches. En primer lugar, AFI asume un detector de congestión basado en el nivel de ocupación del VC, el cual puede ser implementado usando la lógica de encaminamiento actual, mirando los LCs que implementan la política del control de flujo. Específicamente, cuando el nivel de ocupación del VC siguiente alcanza el 75 %, nosotros asumimos que los paquetes mapeados a ese VC contribuyen a la congestión o están afectados.

Respecto a la política de mapeo de flujos a VCs, AFI utiliza una tabla en los LCs (ver el LC del Switch A, puerto de salida 1 en la Figura 4) que relaciona los identificadores de la clase de tráfico (SLs or TCs) con los VCs, dependiendo si el paquete ha sido adaptado o no. Los paquetes que han sido adaptados son mapeados al AFC. Esta tabla es similar a las utilizadas en tecnologías de red comerciales. De hecho, algunos de los SQS descritos en la sección II-C han sido implementados utilizando tablas similares. La única diferencia es que se necesita comparar la cabeza del paquete, que contiene el bit de "adaptado", con la tabla del LC para poder mapear el paquete al VC donde será almacenado en el siguiente switch.

Respecto al algoritmo de encaminamiento, nosotros asumimos el uso D-mod-K para Fat-Trees por defecto cuando la red no está congestionada. Una vez que se detecta congestión en la red, entonces se aplica encaminamiento adaptativo y los flujos de tráfico se puede enviar por cualquier puerto de salida disponible. Respecto al arbitraje en el switch, asumimos una política igualitaria entre los SQS y el AFC, donde todos los VCs tienen la misma prioridad.

V. Evaluación de prestaciones

En esta sección, analizamos el rendimiento de AFI bajo experimentos de simulación, comparando varias propuestas de control de congestión. Hemos llevado a cabo experimentos de simulación en Fat-Trees, bajo tráfico sintético (uniforme y hot-spot), y tráfico basado en la ejecución de aplicaciones reales (mediante archivos de trazas). En esta sección, nosotros describimos los detalles del modelo de simulación usado en nuestra evaluación y comentamos los resultados obtenidos en los experimentos.

A. Modelo de simulación

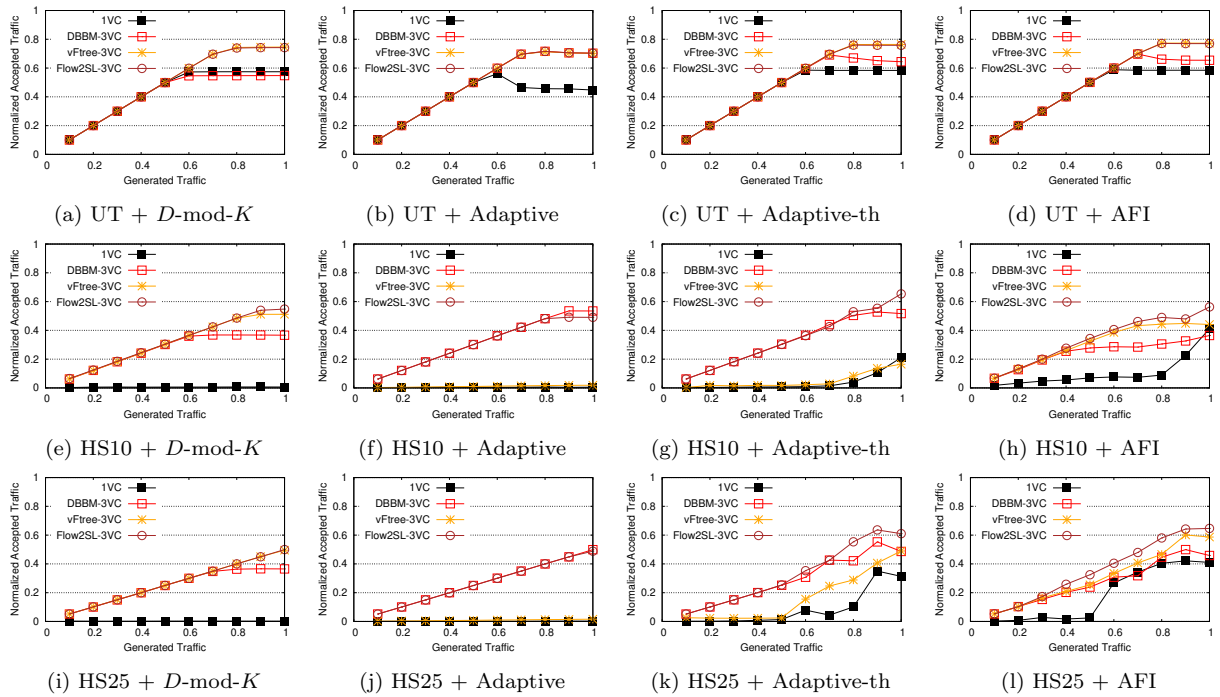
La herramienta de simulación que hemos utilizado en los experimentos es un simulador [22] basado en el framework OMNeT++ [23]. Nosotros hemos implementado en él la técnica AFI para Fat-Trees. La Tabla I muestra las diferentes configuraciones de la topología. Se han modelado dos configuraciones Real Life Fat-Trees (RLFT) con tres etapas, con diferentes tamaños y número de puertos.

TABLA I: Configuración de redes RLFT evaluadas.

# Config.	Nodos	Puertos	Etapas	Switches
1	432	12	3	180
2	11664	36	3	1620

Hemos modelado los algoritmos de encaminamiento descritos en [17], es decir, D-mod-K, adaptativo, adaptativo con umbrales (Adaptive-th) y AFI. Asumimos que AFI usa D-mod-K para rutas deterministas y Adaptive-th cuando adapta flujos de tráfico. Hemos seleccionado un umbral del 75 % para Adaptive-th y AFI. En lo que respecta a la arquitectura de switch, asumimos switches con buffer de entrada (IQ), los cuales pueden ser configurados con VOQs. Asumimos que la política de control de flujo se realiza a nivel de VC, por lo tanto, los flujos de VOQs no tienen control de flujo. Asumimos que el control de flujo está basado en créditos. Hemos modelado AFI para utilizar para poder utilizar cualquier SQSs para comprobar cual funciona mejor junto a nuestra propuesta. Específicamente, hemos modelado Flow2SL, DBBM y vFtree (ver sección II-C) por medio de las siguientes configuraciones de buffer:

- *1VC.* Esta es la configuración de buffer más simple, con sólo 1 VC. Utilizamos este esquema para analizar el la mejora que ofrece AFI cuando no se aplica ningún SQS. En este caso el espacio del buffer se divide en dos VCs (i.e. VC0 and AFC).



(a) UT + D-mod-K (b) UT + Adaptive (c) UT + Adaptive-th (d) UT + AFI

(e) HS10 + D-mod-K (f) HS10 + Adaptive (g) HS10 + Adaptive-th (h) HS10 + AFI

(i) HS25 + D-mod-K (j) HS25 + Adaptive (k) HS25 + Adaptive-th (l) HS25 + AFI

Fig. 5: Productividad Normalizada en función de la carga de Tráfico Generado en un RLFT de 11664-nodos (Config. #2 de la Tabla I), utilizando los algoritmos de encaminamiento D-mod-K, Adaptativo, Adaptive-th y AFI, los esquemas de colas modelados y patrones de tráfico sintético.

- *3VC*(DBBM-3VC, Flow2SL-3VC y vFtree-3VC). SQSs que usan 3 VCs para separar flujos de tráfico cuando estos son "adaptados". Cuando usamos AFI, el espacio del buffer es divido de igual forma entre los 4 VCs tres VCs para SQS y uno para el AFC.

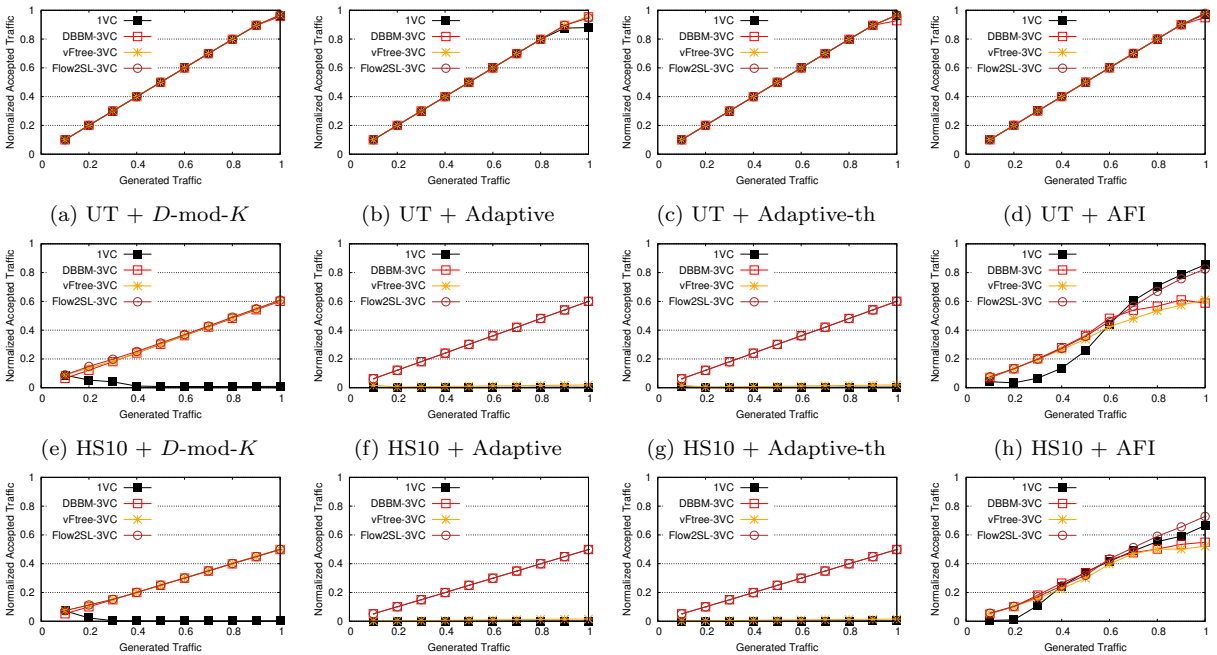
Hemos lanzado experimentos utilizando estas configuraciones en switches IQ con y sin VOQs. El tamaño del buffer de entrada en cada puerto es de 128KB y el tamaño máximo de paquete (MTU) es de 4096 bytes (es decir, se almacenan 32 paquetes por buffer). Asumimos enlaces serie full-duplex con un ancho de banda de 40 Gbps y un retardo de propagación de 6 nanosegundos.

En cuanto a los patrones de tráfico, definimos tres escenarios de tráfico sintéticos diferentes, así como una aplicación basada en trazas reales. El primer patrón de tráfico es el patrón de tráfico uniforme (UT), donde los nodos generan tráfico a destinos aleatorios. El segundo patrón de tráfico (HS10) crea un hot-spot en la red durante todo el tiempo de simulación para modelar un árbol de congestión. Específicamente, en este patrón de tráfico un 10 % de los nodos finales generan paquetes dirigidos a un único destino, mientras que el 90 % de los nodos generan paquetes siguiendo el patrón de tráfico uniforme. El tercer patrón de tráfico (HS25) es también un hot-spot, pero el porcentaje de nodos que generan el hot-spot es del 25 %. Además, se ha modelado el tráfico real gracias al framework de trazas VEF [24]. En particular, utilizamos trazas reales basadas en tráfico de programas MPI obtenidas a partir del test PTRANS, incluido en el benchmark HPCC [25] que genera tráfico entre 432 nodos.

En cuanto a las métricas de rendimiento, medimos el rendimiento normalizado frente a la carga de tráfico (variando la tasa de generación del 0 % al 100 % del ancho de banda máximo del enlace). Finalmente, también proporcionamos el tiempo de ejecución para la aplicación PTRANS.

B. Resultados de tráfico sintético

La Figura 5 y 6 muestran los resultados del experimento para los esquemas de colas modelados con la configuración de red #2 (ver Tabla I), usando arquitecturas de switch sin y con VOQs, respectivamente, y las configuraciones D-mod-k, Adaptative, Adaptive-th y AFI combinados con SQS, bajo patrones de tráfico sintético. La primera fila en la Figura 5 (de la Figura 5a a 5d) muestran los resultados bajo tráfico uniforme(UT). En este caso, el encaminamiento utilizado es indiferente, ya que los periodos de congestión son esporádicos y cortos en el tiempo. Esto se puede observar viendo los resultados de la configuración 1VC que son solo un poco peores cuando se utiliza el encaminamiento Adaptive. La segunda fila de la Figura 5 (de la Figura 5e a 5h) muestra los resultados para un escenario de congestión hotspot (HS10). En este escenario la congestión es fuerte ya que el 10 % de los nodos están generando tráfico direccionado a un único destino. Esto ocasiona que el rendimiento de la configuración 1VC caiga a un valor cercano a 0, y solo reacciona un poco cuando se utilizan las técnicas Adaptive-th o AFI. Lo mismo le sucede a vFtree-3VC cuyo rendimiento mejora significativamente usando AFI (Figura 5h). Además, AFI se sobrepone al problema de diseño de vFtree(la política de mapeo no está pensado para más de dos etapas). Flow2SL y DBBM lidian mejor en este escenario, aunque DBBM no casa muy bien con AFI (obtiene mejores resultados cuando se usa el Adaptativo puro). Flow2SL combinado con AFI logra los



(a) UT + D-mod-K (b) UT + Adaptive (c) UT + Adaptive-th (d) UT + AFI

(e) HS10 + D-mod-K (f) HS10 + Adaptive (g) HS10 + Adaptive-th (h) HS10 + AFI

(i) HS25 + D-mod-K (j) HS25 + Adaptive (k) HS25 + Adaptive-th (l) HS25 + AFI

Fig. 6: Productividad Normalizada en función de la carga de Tráfico Generado en un RLFT de 11664-nodos (Config. #2 de la Tabla I), utilizando los algoritmos de encaminamiento D-mod-K, Adaptativo, Adaptive-th y AFI, los esquemas de colas modelados y patrones de tráfico sintético. Los switches implementan VOQs

mejores resultados en este escenario. Finalmente, en la tercera fila de la Figura 5 (de la Figura 5i a 5l), la congestión se hace incluso más fuerte, donde el 25 % de los nodos generan tráfico a un único destino. Hay un punto interesante en los resultados obtenidos en las Figuras 5k y 5l. En general los resultados se muestran mejores que para el tráfico HS10.

De la misma forma, aunque los mejores resultados son logrados por Flow2SL combinados con AFI, 1VC y vFtree son los que más mejoran al usar AFI. La Figura 6 muestra los mismos escenarios que antes, pero en este caso los switches son modelados utilizando VOQs con el objetivo de prevenir el low-order HoL blocking. Respecto al patrón de tráfico uniforme, los resultados muestran un rendimiento óptimo independientemente del encaminamiento, ya que la arquitectura del switch previene de forma natural el HoL blocking esporádico. Sin embargo, en los patrones de tráfico HS10 y HS25, el HoL blocking generado es más fuerte debido a que el uso de algoritmos adaptativos es contraproducente para las configuraciones de 1VC y vFtree. En general, cuando se utiliza la técnica AFI el rendimiento de la red mejora significativamente para las técnicas 1VC y vFtree, y en menor medida en Flow2SL y DBBM. Por lo tanto, nuestra conclusión es que AFI contribuye significativamente en estas mejoras de rendimiento.

C. *Resultados con trazas basadas en aplicaciones reales*

La figura 7 muestra el tiempo de ejecución (en segundos) de los esquemas de cola modelados combinados con los algoritmos de encaminamiento, usando el test PTRANS como carga de trabajo y la configuración de red #1 (ver Tabla I). En general, podemos observar que las técnicas con configuraciones de buffers que usan VOQs obtienen mejores resultados en términos de tiempo de ejecución (más bajo es mejor) que las que no usan VOQs. El mejor resultado es obtenido por 1VC usando VOQs. AFI funciona un poco mejor que el encaminamiento determinista. También podemos ver que el Adaptive-th y D-mod-K logran los mejores resultados, cuando se combina con esquemas de colas, independientemente de la organización del buffer. Sin embargo, hay que tener en cuenta que esto se debe a que la aplicación PTRANS genera un tráfico uniforme con pequeñas ráfagas de congestión. Es decir, el tráfico de la aplicación genera low-order HoL blocking pero no introduce demasiado high-order HoL blocking.

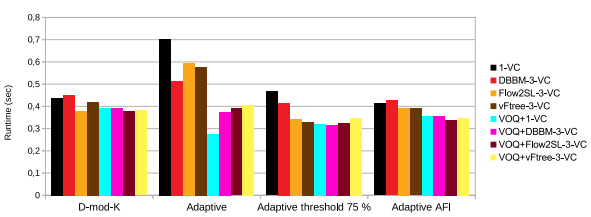


Fig. 7: Tiempo de ejecución de la traza PTRANS.

VI. Conclusiones

En este trabajo hemos descrito la técnica Adapted-Flow Isolation (AFI), que previene del impacto negativo que origina esparcir flujos congestionados sobre varias rutas cuando se utiliza un algoritmo de encaminamiento adaptativo. Esta propuesta utiliza un mapeo dinámico sobre los canales virtuales (VCs) para marginar los flujos adaptados en un VC especial, mientras el resto de flujos permanecen separados en

otros VCs que siguen una asignación estática de colas. Con esta estrategia, nosotros evitamos que los flujos no adaptados sufran HoL blocking debido al esparcimiento de flujos congestionados por la red. Hemos realizado un extenso conjunto de experimentos para probar nuestra propuesta, modelando grandes topologías Real Life Fat-Trees, bajo tráfico sintético (uniforme y hotspot) y trazas reales basadas en tráfico de programas MPI. El análisis de los resultados obtenidos muestra que AFI mejora los esquemas de cola estáticos en los escenarios fuertemente congestionados. Además, en este trabajo definimos detalles para su implementación tales como el detector de congestión, el motor de encaminamiento, y la tabla de mapeo a VCs. Como trabajo futuro, planeamos probar otros mecanismos para detectar la congestión, aunque el mecanismo utilizado es suficiente para probar la eficiencia de la técnica. Además, la tabla de mapeo de VCs permite explorar otras posibilidades para ampliar AFI utilizando más VCs para otras mejoras.

Agradecimientos

Este trabajo ha sido co-financiado conjuntamente por el ministerio de Ciencia, Innovación y Universidades español y la Comisión Europea (fondos FEDER) en el marco del proyecto RTI2018-098156-B-C52, y por la Junta de Comunidades de Castilla-La Mancha en el marco de los proyecto PEII-2014-028-P y SBPLY/17/180501/000498. También está cofinanciado por la Universidad de Castilla-La Mancha y FEDER bajo el marco del proyecto 2019-GRIN-27060. Jesús Escudero-Sahuquillo está financiado por la UCLM y la Comisión Europea (fondos FSE), bajo el programa de investigación de la UCLM (Fecha de resolución UCLM: 31/07/2014).

Referencias

[1] C. E. Leiserson, "Fat-trees: Universal networks for hardware-efficient supercomputing," *IEEE Transactions on Computers*, vol. C-34, no. 10, pp. 892–901, Oct 1985.

[2] Crispín Gómez Requena, Francisco Gilabert Villamón, María Engracia Gómez, Pedro López, and José Duato, "Deterministic versus Adaptive Routing in Fat-Trees," in *Proc. of 21th IEEE IPDPS, Long Beach, California, USA*. Mar. 2007, pp. 1–8, IEEE.

[3] Eitan Zahavi, Greg Johnson, Darren J. Kerbyson, and Michael Lang, "Optimized InfiniBand^{TM} fat-tree routing for shift all-to-all communication patterns," *Journal of CCPE*, vol. 22, no. 2, pp. 217–231, 2010.

[4] Eitan Zahavi, Isaac Keslassy, and Avinoam Kolodny, "Distributed adaptive routing convergence to non-blocking DCN routing assignments," *IEEE Journal on Selected Areas in Communications*, vol. 32, no. 1, pp. 88–101, 2014.

[5] Patrick Geoffray and Torsten Hoefler, "Adaptive routing strategies for modern high performance networks," in *16th Annual IEEE Symposium on High Performance Interconnects (HOTI 2008), 26-28 August 2008, Stanford, CA, USA*. 2008, pp. 165–172, IEEE Computer Society.

[6] John Kim, William J. Dally, and Dennis Abts, "Adaptive routing in high-radix CLOS network," in *Proc. of the ACM/IEEE Conference on Supercomputing, November 11-17, 2006, Tampa, FL, USA*. 2006, p. 92, ACM Press.

[7] Pedro Javier García, Jose Flich, José Duato, Ian Johnson, Francisco J. Quiles, and Finbar Naven, "Dynamic Evolution of Congestion Trees: Analysis and Impact on Switch Arch.," in *Proc of HiPEAC'05, Barcelona, Spain*, pp. 266–285.

[8] M. Jurczyk and T. Schwederski, "Phenomenon of higher order head-of-line blocking in multistage interconnection networks under nonuniform traffic patterns," 1996.

[9] K. Yoshigoe, "Threshold-based exhaustive round-robin for the cicq switch with virtual crosspoint queues," in *2007 IEEE International Conference on Communications*, June 2007, pp. 6325–6329.

[10] William J. Dally and Charles L. Seitz, "Deadlock-Free Message Routing in Multiprocessor Interconnection Networks," *IEEE Transactions on Computers*, vol. C-36, no. 5, pp. 547–553, May 1987.

[11] William Dally, P. Carvey, and L. Dennison, "Architecture of the Avici terabit switch/router," in *6th Hot Interconnects*, 1998, pp. 41–50.

[12] Yuval Tamir and Gregory L. Frazier, "Dynamically-Allocated Multi-Queue Buffers for VLSI Communication Switches," *IEEE Trans. Computers*, vol. 41, no. 6, pp. 725–737, June 1992.

[13] T. Nachiondo, J. Flich, and J. Duato, "Buffer Management Strategies to Reduce HoL Blocking," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 21, no. 6, pp. 739–753, June 2010.

[14] Jesús Escudero-Sahuquillo, Pedro Javier García, Francisco J. Quiles, José Flich, and José Duato, "OBQA: Smart and cost-efficient queue scheme for Head-of-Line blocking elimination in fat-trees," *J. Parallel Distrib. Comput.*, vol. 71, no. 11, pp. 1460–1472, 2011.

[15] Wei Lin Guay, Bartosz Bogdanski, Sven-Arne Reinemo, Olav Lysne, and Tor Skeie, "vFtree - A Fat-Tree Routing Algorithm Using Virtual Lanes to Alleviate Congestion," in *In Proc. of 25th IEEE IPDPS 2011, Anchorage, Alaska, USA*. May 2011, pp. 197–208, IEEE.

[16] Jesús Escudero-Sahuquillo, Pedro Javier García, Francisco J. Quiles, Sven-Arne Reinemo, Tor Skeie, Olav Lysne, and José Duato, "A new proposal to deal with congestion in InfiniBand-based fat-trees," *J. Parallel Distrib. Comput.*, vol. 74, no. 1, pp. 1802–1819, 2014.

[17] Jose Rocher-Gonzalez, Jesús Escudero-Sahuquillo, Pedro Javier García, and Francisco J. Quiles, "On the impact of routing algorithms in the effectiveness of queuing schemes in high-performance interconnection networks," in *In Proc. of 25th IEEE HOTI 2017, Santa Clara, CA, USA,*, pp. 65–72.

[18] Eitan Zahavi, "Fat-trees routing and node ordering providing contention free traffic for MPI global collectives," in *In Proc. of 25th IEEE IPDPS Workshops 2011, Anchorage, Alaska, USA,*. pp. 761–770, IEEE.

[19] Mark J. Karol, Michael G. Hluchyj, and Samuel P. Morgan, "Input versus output queueing on a space-division packet switch," *IEEE Trans. Communications*, vol. 35, no. 12, pp. 1347–1356, 1987.

[20] Alejandro Martínez, Raúl Martínez, Francisco José Alfaro, and José L. Sánchez, "A low-cost strategy to provide full qos support in advanced switching networks," *Journal of Systems Architecture*, vol. 53, no. 7, pp. 355–368, 2007.

[21] Pedro Yébenes, German Maglione Mathey, Jesús Escudero-Sahuquillo, Pedro Javier García, and Francisco J. Quiles, "Modeling a switch architecture with virtual output queues and virtual channels in hpc-systems simulators," in *International Conference on High Performance Computing & Simulation, HPCS 2016, Innsbruck, Austria, July 18-22, 2016*, 2016, pp. 380–386.

[22] Pedro Yebenes, Jesús Escudero-Sahuquillo, Pedro Javier García, and Francisco J. Quiles, "Towards modeling interconnection networks of exascale systems with omnet++," in *21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, PDP 2013, Belfast, United Kingdom, February 27 - March 1, 2013*, 2013, pp. 203–207.

[23] OpenSim Ltd, "OMNeT++ Discrete Event Simulator," http://omnetpp.org/.

[24] Francisco J. Andujar, Juan A. Villar, Francisco J. Alfaro, José L. Sánchez, and Jesús Escudero-Sahuquillo, "An open-source family of tools to reproduce mpi-based workloads in interconnection network simulators," *The Journal of Supercomputing*, vol. 72, no. 12, pp. 4601–4628, 2016.

[25] "The HPCC benchmark," 2017, http://icl.cs.utk.edu/hpcc/.

Metodología de selección de recursos de cómputo para entornos de Cloud Computing

Hugo Haurech[1] y David la Red Martínez[2]

Resumen— **Debido a los avances tecnológicos, las organizaciones se enfrentan a diversos retos al proporcionar el apoyo mediado por las tecnologías de la información (TI), para el desarrollo de las actividades que requieran capacidades de cómputo. Ante el desafío que representa adoptar tecnologías nuevas, resulta imperioso escoger de manera adecuada los recursos para abordar las necesidades de forma eficaz, a fin de alcanzar la capacidad de procesamiento acorde a los requerimientos actuales. La computación en la nube representa una alternativa que ofrece multiplicidad de oportunidades que pueden ser explotadas. Conforme a ello este *paper* presenta las características del *Cloud Computing* (CC), las tecnologías que permiten su despliegue y las herramientas apropiadas para la selección de aquellos recursos.**

Palabras clave— ***Cloud computing*, recursos computacionales, redes computacionales.**

I. Introducción

LOS avances en las áreas de la informática y las comunicaciones están transformando el modo en el cual una organización accede a recursos de TI. El CC representa un nuevo modelo tecnológico a través del cual las organizaciones tienen acceso a una plataforma con características técnicas como la escalabilidad y elasticidad; y funcionales como la eficiencia, por nombrar algunas. De acuerdo a ello se puede definir al CC como un modelo que permite el "acceso en red omnipresente conveniente y bajo demanda" [1], a recursos informáticos que permiten la escalabilidad en forma rápida y con poco esfuerzo. La capacidad de acceso a recursos informáticos es una particularidad que hace atractivo el uso del CC en las organizaciones, debido a que permite alcanzar un conjunto de aplicaciones, servicios y capacidades mediante una red de datos o Internet. De forma similar y con mayor precisión el *National Institute of Standards and Technology* (NIST), lo presenta como "un modelo tecnológico que permite el acceso ubicuo, adaptado y bajo demanda en red a un conjunto compartido de recursos de computación configurables -redes, servidores, equipos de almacenamiento, aplicaciones y servicios-, que pueden ser rápidamente aprovisionados y liberados con un esfuerzo de gestión reducido o interacción mínima con el proveedor del servicio" [2].

Es posible afirmar que la entrega de capacidades de cómputos por medio de recursos virtualizados, mediante una red de datos es una de las características del CC y por ello su uso lo hace conveniente para una organización, donde la infraestructura instalada no se encuentra a la altura de las necesidades de procesamiento para ejecutar cálculos complejos y en particular, de forma paralela. En cuanto a la infraestructura virtualizada para el desarrollo de las actividades, puede ser alcanzada en cualquier momento y lugar, lo que brinda una sensación de independencia al poder utilizar una variedad de dispositivos incluyendo computadoras personales, computadoras portátiles (*Notebooks*), dispositivos inteligentes (*Smartphones, Tablets*), sin necesidad aparente de la instalación de un *software* dedicado. En cuanto al personal de soporte de TI, el CC genera un cambio en el modelo de trabajo particularmente referido a la instalación de herramientas y aplicaciones en los terminales de usuario, que le permite concentrarse en otras funciones, como el funcionamiento correcto de la red de datos y la conectividad a Internet, para que el acceso a los recursos se realice de manera adecuada.

Es posible determinar que este paradigma se ha convertido en los últimos tiempos en una tendencia tecnológica que ha cambiado el modo en que los recursos computacionales son ofrecidos y por consiguiente ha influenciado en el mercado de las TI motivado por la eliminación de las complejas restricciones que supone el entorno informático tradicional, asociadas al espacio, el tiempo, la energía y los costos de equipamiento.

Por lo tanto, para abordar un despliegue en la nube es necesario conocer cuáles son las características del CC, los modelos vigentes, las partes que lo conforman, las tecnologías involucradas y cómo pueden ser seleccionadas de manera adecuada.

Este artículo se organizará de la siguiente manera: en la Sección II se tratará el paradigma del CC, las definiciones consideradas relevantes, sus características y modelos; en la Sección III se abordará la tecnología clave para el desempeño adecuado del CC; en la Sección IV se presentarán las metodologías actuales para la toma de decisión orientada a la selección de recursos; en la Sección V se presentará una propuesta de una metodología de selección; finalizándose con las Conclusiones y líneas futuras de trabajo.

II. Computación en la nube

El paradigma de la computación en la nube implica un cambio en la manera de acceder u ofrecer servicios, aplicaciones e infraestructura, debido a que el acceso a éstos se hace a través de una red de datos o Internet. Según [3] el CC "es la evolución de un conjunto de tecnologías que afecta al enfoque de las or-

[1]Dirección de Tecnologías, Facultad de Ciencias Económicas - UNaM (Posadas, Argentina), e-mail: haurech@fce.unam.edu.ar

[2]Departamento de Informática, Facultad de Ciencias Exactas y Naturales y Agrimensura - UNNE (Corrientes, Argentina), e-mail: lrmdavid@exa.unne.edu.ar

ganizaciones donde un conjunto de *hardware* y *software*, almacenamiento, servicios e interfaces facilitan la entrada de la información como un servicio".

Se desprende de lo anterior que la nube no es un lugar, sino un método de gestión de recursos de TI que reemplaza las máquinas locales y los centros de datos privados con infraestructura virtual. De este modo, los usuarios acceden a los recursos virtuales de computación, red y almacenamiento que están disponibles en línea a través de un proveedor remoto. Además estos recursos pueden aprovisionarse de manera instantánea, lo que es particularmente útil para las organizaciones que necesitan escalar su infraestructura o reducirla rápidamente frente a una demanda fluctuante, situación característica del cómputo en paralelo.

Es posible aquí reconocer la palabra clave **servicios**, los cuales deben dividirse en categorías para determinar por un lado qué servicios se pueden ofrecer, y por otro, de qué manera son ofrecidos. Además, existen características reconocidas como esenciales por el NIST [4], que deben ser tenidas en cuenta al momento de un despliegue en la nube.

Características esenciales

El modelo de CC del NIST proporciona características esenciales que la diferencian de la computación tradicional, con lo cual brinda una base para comparar los servicios en la nube y sus estrategias de implementación. Por lo tanto a partir de la definición proporcionada por el NIST se presentan las cinco características que debe poseer una implementación para ser considerada como CC [4]:

- **Autoservicio bajo demanda**, un consumidor puede aprovisionar unilateralmente capacidades informáticas, según sea necesario de forma automática sin requerir de alguna interacción humana con cada proveedor de servicios. Con ello se refiere a que cualquier usuario de la nube puede tener acceso a los recursos computacionales cuando éste los necesite y sin ningún tipo de interacción con el personal encargado de la nube, de manera automática y unilateral, de manera que logra cierta independencia para la gestión de recursos.
- **Amplio acceso a la red**, las capacidades están disponibles a través de la red y se accede a ellas a través de mecanismos estándar que promueven el uso de plataformas heterogéneas (por ejemplo, teléfonos móviles, tabletas, computadoras portátiles y estaciones de trabajo). Esto garantiza que cualquier usuario, con cualquier sistema operativo o dispositivo (computadores, teléfonos móviles, asistentes personales, etc.) tengan acceso a los servicios.
- **Puesta en común de recursos**, los recursos informáticos del proveedor se agrupan para servir a múltiples consumidores utilizando un modelo de múltiples usuarios, con diferentes recursos físicos y virtuales dinámicamente asignados y reasignados de acuerdo con la demanda del consumidor. Ésto permite a los distintos proveedores compartir sus recursos entre los distintos usuarios, disminuyendo los costes y maximizando la disponibilidad de los mismos.
- **Elasticidad rápida**, las capacidades se pueden aprovisionar y liberar elásticamente, en algunos casos automáticamente, para escalar rápidamente en proporción a la demanda. Indica que los recursos deben ser otorgados según las necesidades del cliente en el momento en que éste los solicite. La adición de recursos se puede dar de dos maneras: horizontalmente (ampliando el número de recursos) o verticalmente (cambiando los actuales recursos por otros con mayores capacidades).
- **Servicio medido**, los sistemas en la nube controlan y optimizan automáticamente el uso de los recursos al aprovechar una capacidad de medición para el tipo de servicio ofrecido (por ejemplo, almacenamiento, procesamiento, ancho de banda y cuentas de usuario activas). El uso de recursos puede ser monitoreado, controlado e informado, proporcionando transparencia tanto para el proveedor como para el consumidor del servicio utilizado. Esto indica que el uso de cualquier recurso debe ser medido, auditado y reportado al cliente en base a un sistema de medición acordado previamente entre el proveedor y el usuario. De esta manera al usuario se le generan cargos económicos según la capacidad o características del servicio contratado.

Modelo de servicio

Se refiere a los "servicios específicos a los que se puede acceder en una plataforma de CC" [3], también conocido como XaaS (*as service* o como servicio) el NIST [4] presenta y define tres modelos estandarizados los cuales son: SaaS (*software as service*), PaaS (*platform as service*) y IaaS (*infrastructure as service*), que presentan las siguientes funcionalidades:

- Desarrollo de aplicaciones y otros servicios.
- Análisis de datos y creación de modelos estadísticos.
- Desarrollo y administración de *software*.
- Almacenamiento, respaldo y recuperación de datos.
- Servicios de *hosting* de sitios *webs*.
- Capacidad de procesamiento.

Programas como servicio (SaaS)

Este servicio otorga al consumidor la capacidad de usar las aplicaciones del proveedor que se ejecutan en una infraestructura en la nube. Se puede acceder a ellas mediante dispositivos (con diferentes tecnologías) a través de una interfaz de cliente, como ser un navegador web (por ejemplo, correo electrónico basado en web o algún portal de usuario). El consumidor no posee la capacidad de administrar ni controlar la infraestructura de nivel inferior o subyacente de la nube como lo son la red, los servidores, los sistemas operativos, el almacenamiento o incluso las ca-

pacidades de aplicaciones individuales. Sin embargo, es posible de manera excepcional el acceso a configuraciones limitadas relacionadas con aplicaciones específicas para el usuario.

Plataforma como servicio (PaaS)

El consumidor posee la capacidad de implementar en la infraestructura de la nube aplicaciones creadas o nuevas, utilizando lenguajes de programación y herramientas compatibles con el proveedor. Éste no posee la capacidad de administrar ni controlar la infraestructura de nivel inferior o subyacente de la nube, dentro de los cuales se encuentran incluidas la red, los servidores, los sistemas operativos o el almacenamiento. Pero tiene control sobre las aplicaciones implementadas y posiblemente las configuraciones del entorno de alojamiento de las mismas.

Infraestructura como servicio (IaaS)

El consumidor posee la capacidad de acceder a procesamiento, almacenamiento, redes y otros recursos informáticos donde puede implementar y ejecutar *software* arbitrario, que puede incluir sistemas operativos y aplicaciones. En este caso, no administra ni controla la infraestructura de nivel inferior, sin embargo tiene el control sobre los sistemas operativos, el almacenamiento, las aplicaciones implementadas y, eventualmente, el control limitado de los componentes de red seleccionados.

Los posibles servicios que se pueden ofrecer y cómo quedan organizados según su taxonomía, se representan en la Figura 1.

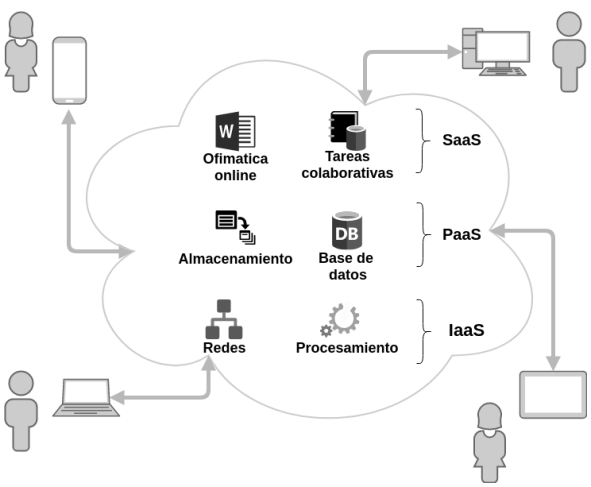


Fig. 1. Representación del modelo de servicios (Fuente propia).

Modelo de despliegue

Según NIST el modelo de despliegue diferencia y define el propósito de la nube y además en dónde se encuentra ubicada. Por lo tanto esta clasificación hace referencia al nivel y tipo de compartición de los recursos contratados en la nube con otras entidades semejantes o de distinta naturaleza.

De acuerdo a ello existen cuatro categorías para gestionar recursos informáticos en la nube, las cuales se presentan a continuación:

Nube privada

En este tipo de modelo la infraestructura de la nube se opera únicamente para una organización, puede ser administrada por la organización o por un tercero y además existir en el local o fuera del mismo.

Se caracteriza por [5]:

- Requerir inversión de capital para la implementación.
- Disponer de un control total de la infraestructura, de los sistemas y de la información.
- Tener un tiempo de respuesta bajo y una alta flexibilidad de asignación de recursos.
- Ofrecer la posibilidad de aprovechar el personal existente y las inversiones realizadas con anterioridad.

Nube pública

La infraestructura en la nube está disponible para el público en general o para un gran grupo industrial y es propiedad de una organización que vende servicios en la nube.

Se caracteriza por [5]:

- Aprovechar la infraestructura de los proveedores de servicios, otorgando una alta escalabilidad y flexibilidad para cambios en el dimensionamiento de servicios.
- Ofrecer el servicio bajo el principio de pago por uso, con lo cual se obtienen costos acordes a lo utilizado.
- Alojar la información corporativa en la nube pública junto a la del resto de clientes del proveedor, lo que implica, además de no poder tener localizada física e ininterrumpidamente dicha información, imponer al proveedor una serie de requisitos de alta exigencia en temas de seguridad y protección de datos.

Nube comunitaria

La infraestructura en la nube es compartida por varias organizaciones cuyas funciones y servicios sean comunes, permitiendo con ello la colaboración entre grupos de interés. Puede ser administrada por las organizaciones o un tercero y puede existir dentro o fuera de los locales.

Se caracteriza por [5]:

- Poseer un número de usuarios menor a los de la nube pública, lo que permite mayores prestaciones en cuestiones de seguridad y privacidad.
- Brindar un conjunto de recursos disponibles mayor respecto a una nube privada, con las ventajas evidentes que ello conlleva en términos de elasticidad.
- Disponer de una cantidad de recursos menor que los existentes en una solución de nube pública, limitando la elasticidad respecto a dicha nube.

Nube híbrida

La infraestructura de nube es una composición de dos o más nubes (privadas, comunitarias o públicas)

que permanecen como entidades únicas pero están unidas por tecnología estandarizada que permite la portabilidad de datos y aplicaciones.

Se caracteriza por [5]:

- Permitir una rápida puesta en servicio.
- Abordar mayor complejidad en la integración de servicios, debido a las diferencias en los tipos de implementación.
- Integrar las mejores características de los distintos tipos de modelos.
- Ofrecer una mayor flexibilidad en la prestación de servicios.

La relación que guardan entre cada modelo queda representada en la Figura 2.

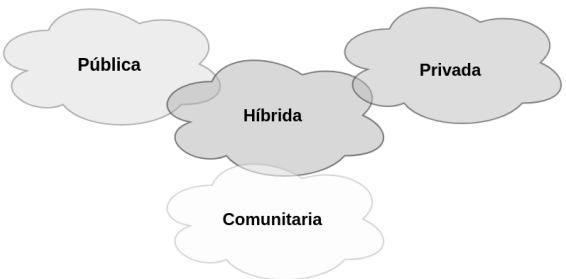


Fig. 2. Representación del modelo de despliegue (Fuente propia).

Agrupamiento de modelos

La clasificación de modelos, de entrega y de servicios, se relaciona con las características esenciales con lo cual es posible ser representada mediante un esquema de bloques de tres capas (Figura 3).

De esta manera la característica esencial de **agrupamiento de recursos**, común a los demás, es la que directamente se relaciona con la capa de modelo de servicios y entrega, por lo que los recursos informáticos se encuentran disponibles para el acceso a múltiples consumidores.

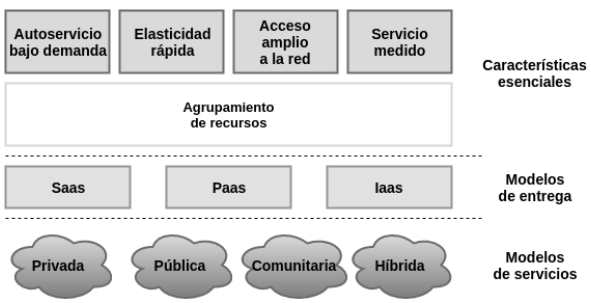


Fig. 3. Representación del modelo de referencia (Fuente propia en base a [4], [6]).

III. Tecnologías de CC

Ante lo expuesto, al CC se lo puede considerar como un sistema distribuido, que consiste en un conjunto de computadoras interconectadas y virtualizadas [7], donde es de resaltar que una herramienta tecnológica como la virtualización se considera una pieza necesaria y fundamental para su despliegue, debido a que mediante ella es posible alcanzar las características de elasticidad y escalabilidad.

Se observa que una infraestructura con esta tecnología sienta las bases para nubes de alto rendimiento, por lo cual se considera una estrategia exitosa para la consolidación de los centros de datos, debido a que logra optimizar los recursos físicos así como también proporcionar los elementos básicos, para que la entrega de servicios en un sistema en nube pueda realizarse de manera ágil y flexible.

En términos de [1], "la virtualización es el proceso de crear la versión virtual de algo". En el contexto de la informática, es la capacidad de crear múltiples instancias de diferentes sistemas operativos, aplicaciones y otros recursos y ejecutarlos en un servidor.

En un modelo de la arquitectura de CC, la capa de infraestructura provee un conjunto de capacidades de almacenamiento y recursos informáticos al dividir los recursos físicos utilizando las tecnologías de virtualización. De este modo el modelo de IaaS tiene la capacidad de otorgar recursos virtualizados para el despliegue de un modelo de PaaS y ésta a su vez brindar los servicios necesarios para la puesta en común de SaaS como se representa en la Figura 4.

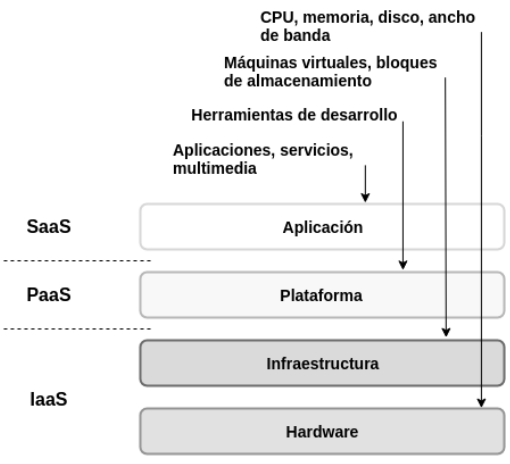


Fig. 4. Representación del modelo de virtualización (Fuente propia en base a [1]).

En el contexto del paralelismo, esta tecnología permite que los servicios alojados en una organización puedan cambiar de ámbito y ser desplegados en máquina virtuales (MV) cuya capacidad de procesamiento, memoria y almacenamiento sean ajustadas de acuerdo a las necesidades conforme a un acuerdo de servicio.

Con ello el principio del paralelismo, que consiste en dividir un problema grande varios problemas pequeños, y posteriormente tratarlos en paralelo pueda llevarse al plano de la virtualización y efectuar el procesamiento de forma simultánea.

Conforme a lo antedicho, las capacidades computacionales necesarias para el procesamiento en paralelo de datos, estaría cubierta por el modelo de servicio de IaaS.

IV. Metodología de selección de recursos

En ciertas oportunidades, suele ocurrir que la selección de recursos para abordar un despliegue en la nube se efectúe de una manera empírica o a través de recomendaciones informales, lo que conlleva a que no siempre se escoja la opción adecuada. Debido a ello se requiere que tanto el personal de trabajo encargado de las TICs, como a sectores claves de una organización deban conocer las alternativas de modelos o procesos para la toma de decisiones y trasladarlas al plano de la selección de recursos, para un abordaje adecuado de la computación en la nube.

Aunque éste sea un acto subjetivo a las preferencias de quienes utilizan algún recurso informático, se debería tener presente una serie de criterios, que permitan afrontar el proceso que implica una toma de decisión de manera adecuada. Resulta indispensable en ese momento contar con la mayor información sobre la organización y las diferentes alternativas posibles a adoptar. Ello reduce la incertidumbre que genera naturalmente este tipo de determinación, en un ámbito de computación en la nube donde existen múltiples opciones disponibles.

De lo expuesto se desprende que resulta necesario contar con una metodología que permita la selección de recursos informáticos, para ser utilizados mediante el modelo de computación en la nube. De acuerdo a este planteo se presentan los modelos utilizados frecuentemente para la toma de decisiones, los cuales consisten en:

- la comparación de recursos.
- el análisis mediante el proceso analítico jerárquico (AHP), que consiste en la presentación de objetivos, criterios y alternativas.
- el análisis mediante el proceso analítico sistémico (ANP), que consiste en la presentación de objetivos, criterios y alternativas, y relación entre los elementos.

A partir de ello se presenta una descripción a fin de conocer cómo operan cada uno.

Modelo comparativo de selección

Este modelo sugiere ser una manera sencilla para la selección de recursos, debido a que simplemente se realiza la comparación entre las herramientas o aplicaciones que se utilizan comúnmente en las actividades y aquellas homólogas que tengan soporte XaaS.

Así, el camino de selección de recursos se puede resumir en los siguientes pasos:

- Conocer el tipo organización y la orientación.
- Identificar las áreas que componen la organización.
- Reconocer las aplicaciones y herramientas tecnológicas que se utilizan para el desarrollo de las actividades.
- Comparar dichas aplicaciones y herramientas con las eventuales soluciones que posee la computación en la nube.

En cuanto al tipo de servicio, es necesario determinar cuál de las alternativas presentadas anteriormente, se ajusta de mejor manera al perfil y actividades de la organización. Por lo tanto el tipo de servicio:

- **SaaS**, será requerido cuando en la organización solamente es necesario contar con aplicaciones *software*, recursos para la compartición de archivos, gestores de contenidos, entornos de aprendizaje, comunicacionales y redes sociales etc.
- **PaaS**, será requerido cuando sea necesario contar con plataformas o herramientas para el desarrollo de *software*, así los miembros de una organización podrán crear, modificar y poner a prueba programas informáticos basados en distintos lenguajes de programación.
- **IaaS**, será requerido cuando sea necesario contar con máquinas virtuales para la puesta en servicio de programas, capacidades de procesamiento, volúmenes de almacenamiento o algún servicio donde sean necesarios recursos virtualizados.

La propuesta puede ser considerada una alternativa válida aunque no adecuada, debido a que la selección queda definida solamente por la comparación de recursos y utilizarla puede presentar inconvenientes al seleccionarlo de manera inadecuada por la falta de criterios de características definidas.

Modelo de selección AHP

Este modelo fue definido por Thomas Saaty [1] quien se caracterizó por realizar un modelo que ayude a afrontar tomas de decisiones complejas. Artículos referidos al tema sugieren emplear técnicas de análisis de resoluciones como el *Analytic Herarchy Process* (AHP), basado en un método cuantitativo con criterios y alternativas que deberán ser incluidos.

Para [9] el AHP es en un método cuantitativo, mediado por criterios y alternativas para apoyar la toma de decisiones que contribuyan a los objetivos de una organización. De manera similar [11] define al método AHP como una propuesta para ordenar el pensamiento analítico, del cual se destacan tres principios básicos: el principio de la construcción de jerarquías, el principio de establecimiento de prioridades y el principio de la consistencia lógica.

Ésto permite dividir un atributo complejo en un conjunto de atributos sencillos y determinar cómo influye cada uno de ellos en el objetivo de la decisión. Esa influencia está representada por la asignación de los valores que se fija a cada atributo o criterio. En resumen, ésto facilita la objetividad del proceso y permite excluir a la intuición en la toma de decisiones.

[1] Thomas L. Saaty: July 18, 1926 – August 14, 2017, distinguido profesor universitario en la Universidad de Pittsburgh, es el inventor, arquitecto y teórico principal del proceso de jerarquía analítica.

Atendiendo la propuesta de [10] es posible establecer un orden de trabajo mediante el modelo AHP, quedando definida la siguiente estructura:

1. La estructuración del modelo jerárquico (objetivos, criterios, alternativas).
2. Priorización de los elementos del modelo jerárquico.
3. Comparaciones binarias entre los elementos.
4. Evaluación de los elementos mediante asignación de pesos (ponderación).
5. Medición de alternativas sujetas a los pesos dados.
6. Síntesis.

En conclusión, parte las etapas que componen el desarrollo de este modelo serían apropiadas, sin embargo la asignación de pesos a los atributos puede introducir errores en la selección, debido a la subjetividad en la aplicación por parte de los decisores.

Modelo de selección ANP

Al igual que el AHP el *Analytic Network Process* (ANP) forma parte de técnicas de decisión multicriterio (TDM), y fue desarrollado con el objetivo de extender las capacidades de AHP a casos en los que existe interdependencia y retroalimentación entre los elementos del sistema [8].

Conforme a [11] este método está dividido en dos partes:

- un control de jerarquía o de red de objetivos y criterios que dominan las interacciones del sistema objeto de estudio.
- el vínculo de las diferentes sub-redes con cada criterio.

La mayor diferencia entre las metodologías de AHP y ANP, es que esta última permite incluir relaciones de interdependencia y realimentación entre elementos del sistema, mientras que los elementos en la metodología AHP son linealmente independientes. Por otro lado, en el método ANP no es necesario establecer diferentes niveles ya que permite obtener una representación del problema de decisión, en un entorno de estructura en red y no una estructura jerárquica [11].

Atendiendo la propuesta de [12] es posible establecer un orden de trabajo mediante el modelo ANP, quedando definida la siguiente estructura:

1. Identificación y agrupación de los criterios de decisión.
2. Establecimiento de los objetivos.
3. Análisis de influencia entre los elementos del sistema (criterios y alternativas).
4. Construcción de la supermatriz original (*unweighted*).
5. Construcción de la supermatriz ponderada (*weighted*).
6. Obtención de la supermatriz límite.

En conclusión, la asignación de pesos a las relaciones entre pares, para representar la intensidad en cada relación, puede introducir inconsistencias asociadas a los juicios de los decisores.

De igual manera que lo analizado en el método anterior, la percepción y valorización de los decisores sobre las asignaciones de pesos, brinda la oportunidad a evaluar nuevas propuestas para la toma de decisión con el fin de minimizar las inconsistencias que podrían generar la selección por medio de AHP y ANP.

V. Propuesta de metodología

Hasta aquí se presentaron tres modelos para la toma de decisiones, sus características singulares, y el proceso para alcanzar la selección. De acuerdo a ello es posible identificar los principales aspectos de cada modelo y presentarlos para que puedan ser evaluados, así:

- **Reconocer el tipo de organización:** es necesario para dar inicio al planteamiento de la solución del problema de la selección de recursos, la información que brinda es de utilidad para reconocer la orientación de la organización. De este modo es posible distinguir qué rasgos deben poseer las herramientas o servicios que utiliza.
- **Identificar las áreas que componen a la organización:** brinda la información necesaria para establecer los recursos que utilizan los usuarios.
- **Comparar los recursos necesarios y disponibles en la nube:** permite identificar cuáles de los recursos requeridos se encuentran disponibles en la nube. Sin embargo esta información no es suficiente para decidir, debido a que puede no cumplimentar en su totalidad con las necesidades de los usuarios, sea en lo funcional o por cuestiones técnicas propias de la computación en la nube.
- **Plantear objetivos, criterios y alternativas:** favorece la interpretación de problema de la selección de recursos, los criterios brindan los parámetros para acotar y filtrar los recursos en la selección, además las alternativas se relacionan con el ítem anterior en cuanto a la búsqueda de recursos.
- **Analizar a través de un proceso matemático:** reviste el conocimiento en áreas diferentes a las de informática las cuales deben ser atendidas por agentes externos a la organización en caso de que el personal de TIC no posea los conocimientos.
- **Establecer un peso a los elementos:** permite asignar un valor ponderable a los elementos (en el caso del modelo AHP) o al vínculo entre ellos (en el caso del modelo ANP). Puede considerarse inadecuado debido a la subjetividad de los decisores o la impericia en cuanto a la aplicación de la metodología.

Conforme a las argumentaciones planteadas y en función de la clasificación propuesta, se considera que aquellos ítems que brindan una calificación po-

sitiva pueden ser candidatos para formar parte de una estructura para modelar la selección de recursos informáticos. Por lo tanto, al realizar un ordenamiento de las actividades queda definida la estructura y puede ser representada en la Figura 5:

- **Identificar el tipo de organización.**
- **Identificar las áreas que componen a la organización.**
- **Plantear objetivos (elementos de la metodología AHP, en el caso de estudio serían los recursos informáticos necesarios).**
- **Establecer los criterios para la selección de los recursos.**
- **La comparación de los recursos necesarios de acuerdo a las alternativas disponibles en la nube que cumplan con los criterios previos.**

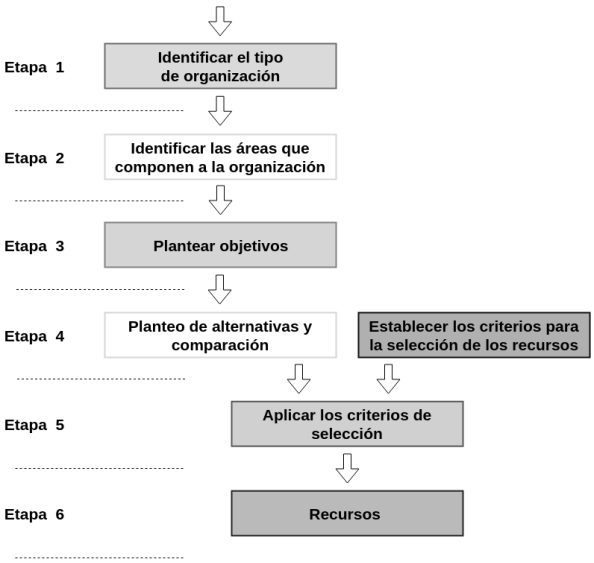


Fig. 5. Representación del modelo propuesto (Fuente propia).

De esta manera la toma de decisión para la selección de recursos que se encuentran disponibles en la nube queda, en si, definida por la identificación de los recursos necesarios y a la definición precisa de los criterios para su adecuada selección.

VI. Conclusiones

La computación en la nube es una tecnología cuya oferta de recursos la hacen muy atractiva para las organizaciones. Los proveedores de esta tecnología ofrecen una variedad de recursos para los modelos de SaaS, PaaS y IaaS, a cada uno de los cuales, de acuerdo al grado de funcionalidad deseada, puede accederse de manera gratuita o de forma remunerada mediante un contrato de servicio.

En cuanto a los recursos que se encuentran disponibles para el modelo de IaaS, la virtualización de servidores y las capacidades de memoria y procesamiento, pueden considerarse como requisitos clave para abordar el cómputo en paralelo con tecnologías de CC. Así, mediante esta tecnología es posible superar las limitaciones de capacidades de cómputo, al contar con una infraestructura con autoservicio bajo demanda y con elasticidad rápida, que ofrece el CC mediante tecnologías de virtualización.

Conforme a lo anterior, escoger recursos en la nube que sean adecuados para la actividad donde sean necesarias capacidades de cómputo para el procesamiento en paralelo, presenta un desafío que los profesionales de las TICs deben abordar. En esta situación se presentan alternativas para la toma de decisión, que al aplicarlas dejan entrever el grado de acierto, la dificultad en su metodología y la subjetividad. No obstante, la síntesis de las mejores actividades que componen las metodologías analizadas, brinda la oportunidad de establecer un método de selección basado en la identificación de los recursos y la definición de criterios precisos, capaz de otorgar una herramienta que supera las limitaciones presentadas.

Avanzando con el tema de estudio, se propone para futuras lineas de investigación introducir a los proveedores de servicios en el análisis de selección, de esta manera se pretende abordar de manera general la inserción de tareas de cómputo en paralelo en ambientes de CC.

Referencias

[1] A. Habbal, S. A. Abdullah, E. O. C. Mkpojiogu, S. Hassan, and N. Benamar, "Assessing Experimental Private Cloud Using Web of System Performance Model," Int. J. Grid High Perform. Comput., vol. 9, no. 2, pp. 21–35, Apr. 2017.

[2] P. Mell and T. Grance, "The NIST Definition of Cloud Computing Recommendations of the National Institute of Standards and Technology," Nist Spec. Publ., vol. 145, p. 7, 2011.

[3] L. Joyanes Aguilar, "Computación en la Nube. Notas para una estrategia española en cloud computing," Rev. del Inst. Español Estud. Estratégicos, pp. 89–112, 2012.

[4] L. Badger et al., "US Government Cloud Computing Technology Roadmap," Nist Spec. Publ., vol. I; II, p. 85, 2014.

[5] S. Carlos, B. Guzmán, and M. S. Rodríguez, "Computación en la nube, una tecnología emergente en la educación y en el sector empresarial: beneficios y desventajas desde el punto de vista operativo y ambiental," Revista Iberoamericana para la Investigación y el Desarrollo Educativo, pp. 1-42, 2007.

[6] R. B. Bohn, J. Messina, F. Liu, J. Tong, and J. Mao, "NIST cloud computing reference architecture," Proc. - 2011 IEEE World Congr. Serv. Serv. 2011, pp. 594–596, 2011.

[7] H. Chihi, W. Chainbi, and K. Ghdira, "Cloud computing architecture and migration strategy for universities and higher education," Proc. IEEE/ACS Int. Conf. Comput. Syst. Appl. AICCSA, vol. 2016-July, 2016.

[8] J. María Moreno-Jiménez, "Selección Multicriterio de un Sistema Erp Mediante Las Metodologías Ahp Y Anp," Congreso de Logística y Gestión de la Cadena de Suministros, 2007.

[9] P. Medina, E. Cruz, and R. Gomez, "Selección de proveedor de WMS utilizando método AHP," Sci. Tech., vol. 17, no. 52, pp. 65–72, 2012.

[10] C. A. Bermubez Irreño and E. D. Quiñonez Aguilar, "Aplicación Práctica Del Proceso De Análisis Jerárquico (Ahp), Para La Toma De Decisiones," Rev. Ing. Matemáticas y Ciencias la Inf., vol. 5, no. 9, pp. 91–100, 2018.

[11] G. C. Guerrero-liquet and J. Faxas-guzmán, "Análisis de toma de decisión con AHP / ANP de energías renovables en República Dominicana República Dominicana," Anuario de Jóvenes Investigadores, vol. 8, pp. 27–29, 2015.

[12] A. Sampedro-durá, I. Puchol-garcía, and P. Aragonésbeltrán, "Aplicación del proceso analítico en red ANP para la selección de un project manager," XV Congreso Internacional de Ingeniería de Proyectos, pp. 6–8, 2011.

Diseño de una aplicación de mensajería tolerante a desconexión

Juan Cúñez-Olalla, Jefferson Rodríguez-Sánchez, Jorge Herrera-Tapia, [1] Leonardo Chancay-García, Enrique Hernández-Orallo, Carlos Tavares Calafate, Juan-Carlos Cano, Pietro Manzoni [2]

Resumen— **Compartir información o simplemente mantenerse comunicado se ha vuelto necesario, por lo general a través de aplicaciones de mensajería que utilizan Internet en los teléfonos inteligentes. Sin embargo, esta comunicación es difícil en lugares donde no existe infraestructura de telecomunicaciones y los canales de transmisión se encuentran saturados por las innumerables peticiones de conexión. En este contexto, la utilización de redes oportunistas sería una alternativa para que las personas puedan comunicarse. Este tipo de redes aprovecha la oportunidad de contacto entre dispositivos móviles, siempre y cuando los usuarios estén dispuestos a colaborar. En este artículo se presenta una aplicación de mensajería tolerante a desconexión, basada en redes oportunistas, utilizando WiFi-Direct, que permite el envío de mensajes de texto y multimedia. La evaluación se realizó sobre el tiempo de transmisión para cada tipo de mensaje con mediciones outdoor e indoor entre los dispositivos, además se realizaron simulaciones, comprobando de esta manera la factibilidad de uso de las redes oportunistas en los dispositivos inteligentes.**

Palabras clave— **Redes Ad-hoc, WiFi-Direct, mensajería instantánea, redes oportunistas.**

I. Introducción

Los sistemas de mensajería instantánea por su versatilidad son la aplicación más utilizada para la comunicación entre personas, y utilizan la infraestructura de Internet para su funcionamiento. Existen escenarios donde esta comunicación no es posible o su infraestructura de transmisión de datos no se encuentra en condiciones de funcionar de forma apropiada, como es el caso de localidades que han sido impactadas por algún desastre natural, o en determinados eventos donde existe masificación de gente y la infraestructura de telecomunicaciones se encuentra saturada debido a las múltiples peticiones de conexión de los los usuarios.

Los mensajes que envían las personas varían de acuerdo a la circunstancias, pudiendo ser mensajes de texto cortos, concernientes a una petición, alerta o sugerencia, así como fotos y vídeos que evidencien determinadas situaciones, por ejemplo una feria o concierto.

En este trabajo se presenta el diseño de una aplicación de mensajería tolerante a desconexión para dispositivos con sistema operativo Android, capaz de mantener los mensajes en memoria si los dispositivos perdieran conexión, y que trate de enviar la información utilizando dispositivos intermediarios hasta que los mensajes lleguen al destino. Se presentan resultados acerca del rango de transmisión utilizando WiFi en un escenario abierto sin obstáculos, y en un escenario cerrado con obstáculos. Para determinar la factibilidad de comunicación a través de este tipo de redes inalámbricas se procedió a analizar la difusión de mensajes utilizando simulaciones con el simulador The ONE (The Opportunistic Network Environment) [1].

Este sistema de comunicación que se propone, está basado en redes oportunistas. Los autores de [2–4], en su trabajo explican a detalle el funcionamiento de este tipo de redes, basadas en la oportunidad de transmitir información cuando los dispositivos establecen contacto entre ellos. El tiempo de transmisión de los datos está relacionado con el tamaño y la distancia existente entre los dispositivos o nodos que actúan como emisor y receptor. En base a esto se ha realizado la evaluación de la propuesta.

Este artículo está organizado de la siguiente manera: en la Sección 2 se presenta una breve descripción de las redes oportunistas y de los protocolos de encaminamiento. En la Sección 3 se explica la propuesta de la aplicación diseñada, y en las Secciones 4 y 5 la evaluación y conclusiones respectivamente.

II. Trabajos relacionados

Las redes oportunistas [2–4] por su modelo de difusión son consideradas como una subclase de las Redes Tolerantes a Retraso DTN (Delay Tolerant Networks), los autores de [5–7], ofrecen un amplia descripción y taxonomía de este tipo de comunicación. Las DTN se basan en el principio de "almacenar, transportar y reenviar" los mensajes que reciben los nodos cuando entran en contacto con otros.

El rendimiento de las redes oportunistas depende del número, movilidad y nivel de colaboración de los usuarios de dispositivos móviles, lo que determina la duración del contacto entre los nodos, influyendo en la cantidad de datos transmitidos, así como también del tipo de radio o interfase inalámbrica utilizada en los nodos. Con respecto a esto, los autores de [8, 9] analizaron la difusión de mensajes considerando el tiempo de duración del contacto de acuerdo a la actitud de colaboración o amistad de los usuarios.

Acerca de la aplicación de las redes oportunistas, los autores de [10, 11] exponen los escenarios donde sería factible su despliegue, desde ambientes laborales, comerciales hasta la recolección de datos de re-

[1]Facultad de Ciencias Informáticas, Univ. Laica Eloy Alfaro de Manabí, e-mail: {e1316459088, e1313689265}@live.uleam.edu.ec, jorge.herrera@uleam.edu.ec.

[2]Dpto. de Informática de Sistemas y Computadores, Universitat Politècnica de València, e-mail: leochaga@doctor.upv.es, {ehernandez, calafate, jucano, pmanzoni}@disca.upv.es.

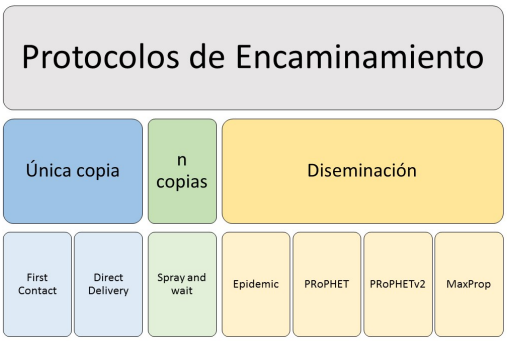


Fig. 1: Clasificación de los protocolos de enrutamiento.

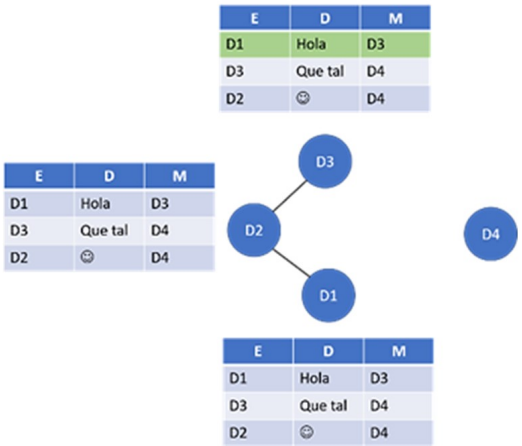


Fig. 2: Difusión epidémica de un mensaje a un usuario determinado.

des de sensores. Los experimentos fueron realizados en escenarios reales durante varias semanas, estableciendo el término de *distanciasocial*. En cambio los autores de [12, 13] presentan a las redes oportunistas como una alternativa válida de comunicación en tiempos de crisis, evaluando la difusión de mensajes de acuerdo a su tamaño y cantidad de nodos.

El rendimiento de las redes oportunistas también es evaluado de acuerdo a la movilidad de los usuarios. Los autores de [14,15] realizaron experimentos donde analizaron la movilidad de acuerdo a determinados patrones de desplazamiento de las personas, como eventos sociales y celebraciones donde se concentra una importante cantidad de personas. Para examinar el rendimiento de las redes oportunistas utilizando simulaciones, los autores de [16] en su trabajo explican en profundidad como ejecutar y analizar este tipo de redes utilizando diferentes herramientas.

Se han propuesto muchos protocolos para la difusión de información en redes oportunistas. La Figura 1 muestra los principales algoritmos de encaminamiento, de acuerdo al número de copias del mensaje a ser transmitidos. De estos protocolos, el más utilizado es el Epidémico, que fundamenta su difusión de mensajes en el efecto de una enfermedad viral. Autores como [11, 17, 18] en sus trabajos evalúan el rendimiento de este algoritmo, que se ha convertido en la base de otros algoritmos de encaminamiento.

Actualmente existen algunas aplicaciones desarrolladas para enviar mensajes sin la utilización de Internet, como por ejemplo FireChat [19] y GRChat [20], entre otras. Firechat fue utilizada por manifestantes en Hong Kong en 2014, la gente utilizaba como medio de comunicación en un escenario donde estaba restringido el uso de Internet. En cambio GRChat, fue desarrollada con el objetivo de evaluar los sistemas de mensajería basadas en contacto.

Los trabajos de investigación citados sitúan a las redes oportunistas como una alternativa viable para la transmisión de mensajes en lugares donde existan usuarios de dispositivos móviles, y que no tengan acceso a Internet, aunque algunas de estas han quedado como simples modelos de difusión. La propuesta que presentamos hace uso de las ventajas de las redes oportunistas combinada con las propiedades de WiFi que disponen los dispositivos celulares.

III. Propuesta

A continuación se explicarán los aspectos más relevantes de la aplicación desarrollada, que se le ha dado el nombre de *ChatInmediato*.

Para implementar el prototipo funcional del sistema se utilizaron las siguientes herramientas de software y consideraciones técnicas: a) Sistema operativo: Android desde versión 4.4 en adelante. b) IDE a usar: Android estudio 3.0. c) Base de datos: SQLITE. d) Método de conexión: WiFi-Direct. e) Tipo de diseminación: Epidémico.

A. Modelo de difusión.

Considerando el funcionamiento de las redes oportunistas, se propone el diseño e implementación de un sistema de mensajería entre dispositivos móviles que soporte la desconexión de estos, sin que se pierdan los mensajes que los usuarios quieran enviar cuando estaban en contacto; finalizando el envío cuando los dispositivos de los usuarios entren en un rango de conexión que soporte la transmisión directa de datos o a través de otros, siempre que estén dispuestos a colaborar con la retransmisión de los mensajes, produciéndose la diseminación epidémica de los mensajes.

Conociendo las propiedades de transmisión de WiFi, especialmente su distancia de transmisión, el sistema utilizará este estándar pero en modo de conexión ad-hoc a través de WiFi-Direct, ampliando de esta manera el rango de conexión y tiempo de contacto de los dispositivos.

El enrutamiento epidémico es una de las formas más eficientes utilizadas para la difusión de mensajes. Autores como [21] afirman que el funcionamiento de este enrutamiento está basado en que un nodo envía el mismo paquete de datos a todos los demás nodos conectados a la misma red. Este paquete estará disponible para ser enviado a cualquier nuevo dispositivo que se conecte y este dispositivo se encargará de distribuir el paquete a otros dispositivos

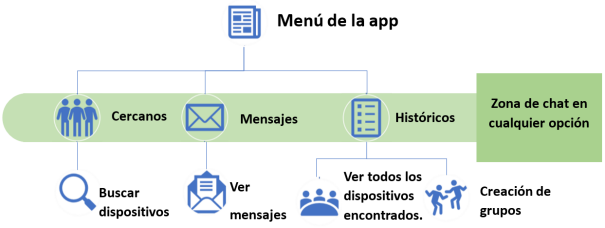


Fig. 3: Funcionamiento de la aplicación.

que establezcan una conexión (ver Figura 2). Solo el nodo destino podrá acceder a esa información, a no ser que el el usuario emisor indique que el mensaje sea para todos.

En el ejemplo de la Figura 2 podemos ver que $D1$ es el emisor del mensaje, con destino $D3$, el mismo que ha utilizado a otros dispositivos como medio de propagación, hasta llegar a su destino. Este es la idea principal del funcionamiento de la propuesta de este proyecto.

B. Funcionamiento de la aplicación.

La Figura 3, muestra la funcionalidad de la aplicación, desde el descubrimiento de dispositivos, la edición de mensajes, la gestión de grupos de destinatarios entre otras opciones.

Una vez instalada la aplicación se escribe el *Nickname* por el cual podrá ser reconocido el usuario, y se empieza la búsqueda de otros dispositivos. Aparece una lista de los dispositivos cercanos, (ver Figura 4). Se envía una petición de conexión con el equipo *remoto*, y si se acepta, se establece la conexión.

Fig. 4: Descubriendo dispositivos en el rango de comunicación.

El equipo *remoto* recibe del *solicitante* todos los mensajes que este tiene alojado en su memoria, pero el usuario solo podrá ver aquellos de los cuales sea el destinatario. El dispositivo siempre está en modo de descubrimiento de otros cercanos con el objetivos de transmitir los mensajes.

El equipo *remoto* envía todos los paquetes de mensajes a los *equipos* con que se conecte. Los dispositivos eliminan de su archivo todos los mensajes cuyo tiempo de vida se ha agotado. El dispositivo envía como paquete el listado de todos los demás dispositivos cercanos que ha encontrado.

La Figura 5, muestra el *chat* entre dos usuarios. La aplicación también sirve para enviar archivos de diferente tipo como audio, dibujos, imágenes y vídeos. La Figura 6 muestra esta funcionalidad, que se activa en la *Sala de chat* de los usuarios.

En resumen, en esta sección se ha explicado las principales opciones de la aplicación. Cabe resaltar que aparte de la interfase de envío y recepción de mensajes entre dispositivos conectados con WiFi-Direct, la aplicación está diseñada para la transmisión de información de manera que tolere la desconexión temporal entre los dispositivos móviles de los usuarios, haciendo que no se pierdan los datos, y que utilice a otros dispositivos como medio de difusión epidémica de información.

IV. Evaluación

En primer lugar se comprobó la funcionalidad de la aplicación, que cumpla con su objetivo de gestionar mensajes cuando los dispositivos móviles no dispongan de Internet. Permitiendo que los mensajes se guarden en memoria hasta que el móvil entre en contacto con otros dispositivos para proceder al proceso de envío. Además se verificó que los nodos ayuden en la difusión de los mensajes, incluso si estos no son el destino final.

Aparte de verificar el funcionamiento de la aplicación, procedimos a realizar mediciones del tiempo de transferencia de diferentes archivos en tamaño y tipo. Se obtuvieron valores promedio de 16 mediciones de cada transmisión, tanto en un ambiente abierto con punto de vista entre los móviles (outdoor) y dentro de un edificio con obstáculos entre los dispositivos (indoor). Se Utilizó WiFi-Direct en la frecuencia de 2.4GHz, con teléfonos inteligentes Samsung modelo J5 PRO-J530G-DS.

La Tabla I muestra los resultados obtenidos en campo abierto (outdoor). Es evidente que entre mayor es el tamaño del archivo, y la distancia entre nodos, el tiempo de transferencia se incrementa. Como se puede ver, el alcance con WiFi-Direct con línea de vista superó los 300 *metros* (m). Transmitir una cadena de caracteres de $32B$ a $18m$ toma 42 *milisegundos* (ms), (es casi instantáneo), así mis-

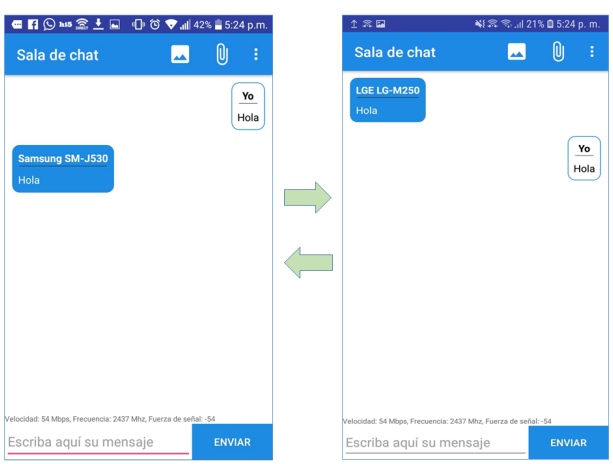


Fig. 5: Ejemplo de chat entre dos usuarios.

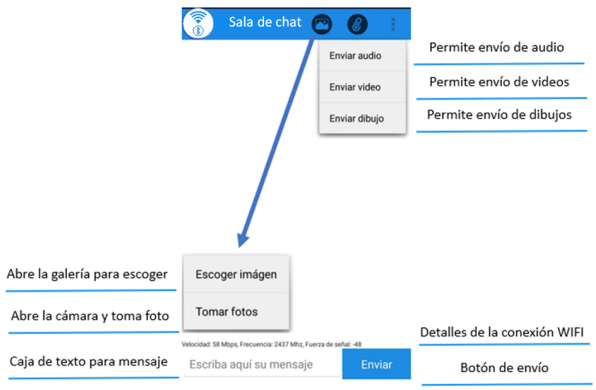


Fig. 6: Enviando archivos multimedia.

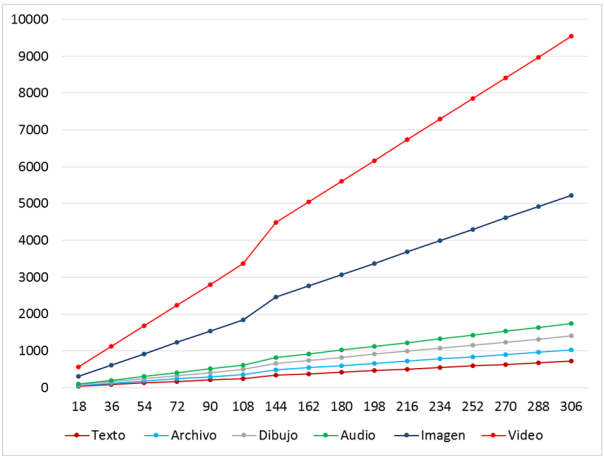


Fig. 7: Tiempo de transferencias de archivos, utilizando la Fórmula 1, eje y en milisegundos, eje x distancia en metros. Mediciones a campo abierto (outdoor).

mo vemos que un archivo de $8kB$ (imagen de baja resolución) o audio toma un poco más de $500ms$. Y transferir un archivo de vídeo corto se demora casi 10 *segundos* a una distancia de $306m$.

En base a los valores obtenidos de las mediciones reales, se obtuvo la Fórmula 1.

$$t = (398 + x) * \frac{d}{183 * (10^n)} \quad (1)$$

Donde:

- t es el tiempo de transferencia.
- x es el peso en bytes del archivo.
- 398 es una constante de metadata.
- 183 es un valor constante de suavizamiento.
- d es la distancia existente entre dispositivos.
- El valor de n varía de acuerdo con el tamaño del archivo a enviar:
  - $n = 0 : 1B - 1kB$
  - $n = 1 : 1kB - 100kB$
  - $n = 2 : 100kB - 1MB$

La Figura 7, muestra los valores al aplicar la fórmula.

En los resultados podemos notar otro aspecto importante de las redes inalámbricas: debido a su naturaleza, es difícil controlar la interferencia de factores invisibles, a veces a favor o en contra, por ejemplo en el gráfico podemos ver que a los $162m$ de distancia el tiempo de transmisión baja, cuando se supone que debe seguir un patrón ascendente. Y esta irregularidad se observa en el resto de mediciones con más o menos incidencia.

Acerca de las mediciones en ambiente indoor. Se consideraron a las paredes como obstáculos entre los dispositivos móviles, el primer obstáculo fue una pared de bloques de cemento (22cm de espesor) con una distancia de $3m$ entre los dispositivos; luego se consideraron 2 paredes a una distancia de $5m$ entre los dispositivos, y finalmente a $9m$ con 3 paredes

TABLA I: Mediciones reales en milisegundos (campo abierto).

Tipo	Texto	Archivo	Dibujo	Audio	Imagen	Vídeo
Dist (m)	32B	215B	8kB	10kB	312kB	570kB
18	42	70	100	108	324	645
36	98	100	140	212	749	1183
54	130	190	260	314	1023	1893
72	196	205	342	402	1229	2130
90	204	280	422	498	1536	2805
108	376	324	540	616	1905	3462
144	338	444	679	876	2644	4472
162	394	583	765	910	2564	3618
180	454	503	900	1001	3124	5794
198	490	679	987	1027	3678	6172
216	524	752	992	1345	3687	6453
234	567	777	1098	1452	3995	7722
252	600	893	1192	1486	4302	8165
270	612	908	1200	1578	4722	8176
288	645	985	1392	1669	5006	8899
306	714	1046	1720	1818	6593	9782

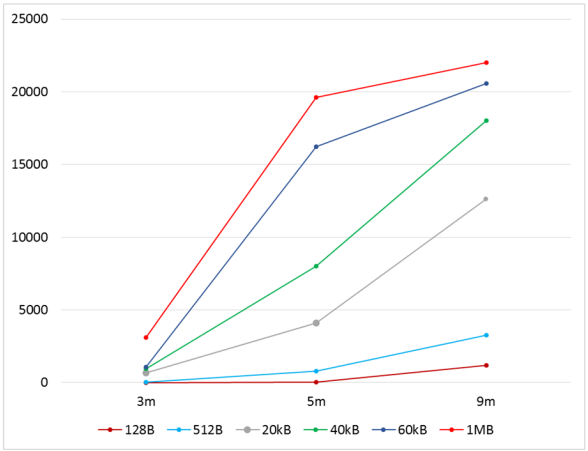


Fig. 8: Tiempo de transferencia de diferentes tipos de archivos, eje y en milisegundos, eje x distancia en metros. Mediciones en un ambiente interior (indoor).

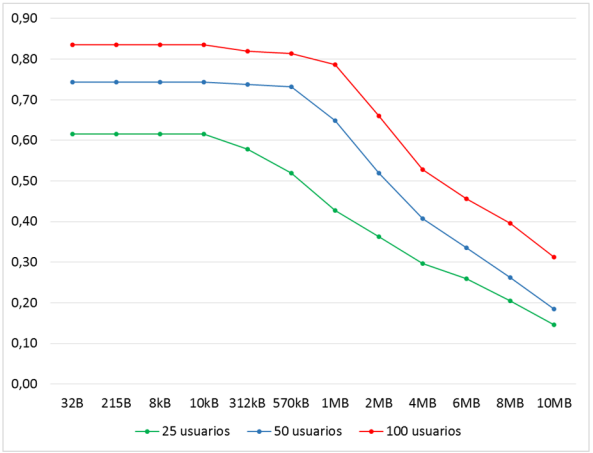


Fig. 9: Probabilidad de entrega de los mensajes de acuerdo a su tamaño y al número de usuarios.

entre los móviles. Además de las paredes había un promedio de 14 personas que circulaban en dichas áreas. La Figura 8 muestra los resultados. Es evidente que los obstáculos si interfieren en la comunicación inalámbrica, por ejemplo en transmitirse un mensaje de $20kB$ con un pared en el medio se demora cerca de $0,5s$, mientras que con 3 *obstáculos* pasan los $12s$.

Para evaluar la factibilidad de la aplicación y de la difusión de mensajes, procedimos a realizar simulaciones utilizando el simulador The ONE (The Opportunistic Network Environment). Los principales parámetros de simulación se muestran en la Tabla II, el escenario es una ciudad, el tiempo de simulación corresponde a 6 *horas*, con una velocidad de los usuarios entre $0,25m/s - 1,5m/s$. Los mensajes se generan con una frecuencia comprendida entre $1 - 3$ *minutos*.

Debido a la actual capacidad de memoria de los teléfonos, se considera el valor del buffer como ilimitado. De acuerdo a las especificaciones el rango de transmisión de WiFi-Direct es de $100m$ y un ancho de banda de $54Mbps$. Pero, en este caso, para obtener resultados más reales, y de acuerdo a los experimentos descritos, vamos a considerar en la simulación valores más bajos: un rango de $30m$ y un ancho de banda de $2Mbps$. por posibles efecto de interferencia y obstáculos. Con el objetivo de conocer el grado y tiempo de diseminación de los mensajes, durante el proceso de simulación iremos variando el número de usuarios y el tamaño de los mensajes que se diseminan en la red. A continuación explicamos los resultados.

TABLA II: Parámetros de Simulación.

Parámetro	Valor
No. usuarios	25, 50, 100
Rango Tx	30 metros
Bandwidth	2Mbps
Protocolo	Epidémico
Mensajes	32B, 215B, (8,10,312,570)kB, (1,2,4,6,8,10)MB

La Figura 9, muestra la probabilidad de entrega de los mensajes generados. Cuando se tiene 100 usuarios o nodos participando en la red, la posibilidad de entrega es cerca del 85 %, manteniéndose hasta con mensajes relativamente pequeños de $10kB$. Conforme va incrementándose el tamaño de los mensajes, la probabilidad va decreciendo, llegando hasta casi un 30 %. Esto se debe a que el tiempo de contacto entre nodos no es lo suficiente para culminar la transferencia de los mensajes grandes. Este efecto de descenso es similar en los casos de 50 y 25 nodos, con una diferencia de aproximadamente 10 % entre estos.

Respecto al tiempo de entrega de los mensajes, en la Figura 10 se presentan los resultados. el efecto del número de usuarios y del tamaño de los mensajes es contrario a la probabilidad de entrega. Se puede apreciar que a un menor número de nodos el tiempo de entrega aumenta, y el efecto se mantiene cuando

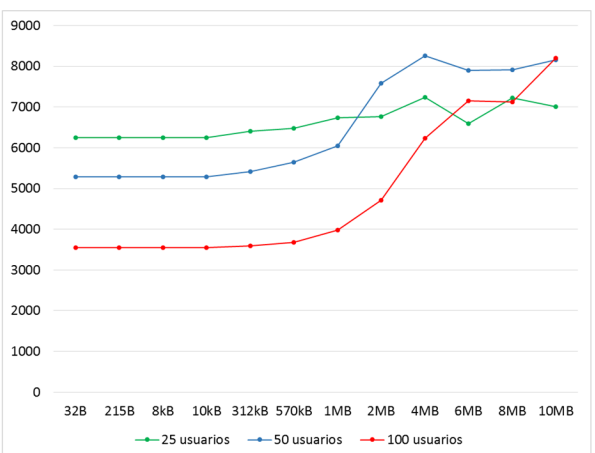


Fig. 10: Tiempo (segundos) promedio de entrega de los mensajes (Latencia), de acuerdo a su tamaño y al número de usuarios.

se incrementa el tamaño de los mensajes. Por ejemplo, para que un mensaje corto de texto se disemine en la red se necesita alrededor de $6300s$, es decir mas de $1,75$ *horas*, mientras que con 100 usuarios, este se diseminaría en aproximadamente $3600s$ (1 hora). Y la tendencia se mantiene con el resto de mensajes y usuarios.

V. Conclusiones

En este trabajo se presentó el diseño y funcionalidad de una aplicación de mensajería instantánea, que soporta la desconexión temporal entre dispositivos, basada en el modelo de redes oportunistas. De acuerdo a las pruebas de campo realizadas se evidencia que la distancia y el escenario influyen en la capacidad de transmisión de la información, y que existen fluctuaciones durante el proceso de transmisión. Los resultados de las simulaciones muestran que el número de usuarios y el tamaño de los mensajes afectan a la difusión de la información, pero en general, si es factible el uso de las redes oportunistas, convirtiéndose estas en una alternativa viable para compartir información a través de mensajes en lugares donde no se cuente con una infraestructura de telecomunicaciones e Internet.

Como trabajo futuro se prevé la implementación de un sistema que utilice las diferentes interfaces de un dispositivo móvil dependiendo del lugar donde se encuentre el usuario. Así mismo analizar mecanismos de gestión de memoria para optimizar la entrega de mensajes.

Agradecimientos

Agradecemos de manera especial a la *Universidad Laica Eloy Alfaro de Manabí*, a través del *Proyecto de Investigación de redes para la transmisión de datos en dispositivos móviles en situaciones emergentes*, perteneciente a la Facultad de Ciencias Informáticas. De igual manera al Grupo de Redes de Computadores (GRC) de la Universitat Politècnica de València, Valencia, España. Este trabajo fue parcialmente financiado por el *Ministerio de Ciencia, Innovación y Universidades, Programa Estatal de Investigación, Desarrollo e Innovación Orientada a los Retos de la Sociedad, Proyectos I+D+I 2018*, España, bajo la subvención RTI2018-096384-B-I00.

Referencias

[1] A. Keränen, J. Ott, and T. Kärkkäinen, "The ONE simulator for DTN protocol evaluation," *Proceedings of the Second International ICST Conference on Simulation Tools and Techniques*, 2009.

[2] L. Pelusi, A. Passarella, and M. Conti, "Opportunistic Networking : Data Forwarding in Disconnected Mobile Ad Hoc Networks," pp. 134–141, 2006.

[3] S. Ferretti, "Shaping opportunistic networks," *Computer Communications*, vol. 36, pp. 481–503, 2013.

[4] V. F. Mota, F. D. Cunha, D. F. Macedo, J. M. Nogueira, and A. A. Loureiro, "Protocols, mobility models and tools in opportunistic networks: A survey," *Computer Communications*, vol. 48, pp. 5 – 19, 2014. Opportunistic networks.

[5] S. Tornell, C. Calafate, J.-C. Cano, and P. Manzoni, "DTN Protocols for Vehicular Networks: an Application Oriented Overview," *IEEE Communications Surveys & Tutorials*, pp. 1–1, 2015.

[6] S. Cha, E. Talipov, and H. Cha, "Data delivery scheme for intermittently connected mobile sensor networks," *Computer Communications*, vol. 36, pp. 504–519, 2013.

[7] M. Ito, H. Nishiyama, and N. Kato, "A novel communication mode selection technique for DTN over MANET architecture," *2014 International Conference on Computing, Networking and Communications (ICNC)*, pp. 551–555, 2014.

[8] J. Herrera-Tapia, E. Hernández-Orallo, A. Tomás, P. Manzoni, C. Tavares Calafate, and J.-C. Cano, "Friendly-sharing: Improving the performance of city sensoring through contact-based messaging applications," *Sensors*, vol. 16, no. 9, p. 1523, 2016.

[9] K. Thilakarathna, A. C. Viana, A. Seneviratne, and H. Petander, "Mobile social networking through friend-to-friend opportunistic content dissemination," *Proceedings of the fourteenth ACM international symposium on Mobile ad hoc networking and computing - MobiHoc '13*, p. 263, 2013.

[10] J. M. Cabero, V. Molina, I. Urteaga, F. Liberal, and J. L. Martín, "Acquisition of human traces with Bluetooth technology: Challenges and proposals," *Ad Hoc Networks*, vol. 12, pp. 2–16, jan 2014.

[11] A. Förster, K. Garg, H. A. Nguyen, and S. Giordano, "On Context Awareness and Social Distance in Human Mobility Traces Categories and Subject Descriptors," pp. 5–12, 2012.

[12] M. Avvenuti, P. Corsini, P. Masci, and A. Vecchio, "Opportunistic computing for wireless sensor networks," *Mobile Adhoc and Sensor Systems. MASS 2007. IEEE International Conference on*, pp. 1–6, 2007.

[13] A. Martín-Campillo, J. Crowcroft, E. Yoneki, and R. Martí, "Evaluating opportunistic networks in disaster scenarios," *Journal of Network and Computer Applications*, vol. 36, pp. 870–880, mar 2013.

[14] L. Chancay-Garcia, J. Herrera-Tapia, P. Manzoni, E. Hernandez-Orallo, C. T. Calafate, and J.-C. Cano, "Evaluation of Routing Protocols for Opportunistic Networks in Scenarios with High Degree of People Renewal," pp. 228–235, may 2018.

[15] J. Herrera-Tapia, E. Hernandez-Orallo, P. Manzoni, A. Tomas, C. T. Calafate, and J.-C. Cano, "Evaluating the Impact of Data Transfer Time and Mobility Patterns in Opportunistic Networks," *2016 Intl IEEE Conferences on Ubiquitous Intelligence & Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People, and Smart World Congress (UIC/ATC/ScalCom/CBDCom/IoP/SmartWorld)*, pp. 25–32, 2016.

[16] J. Dede, A. Förster, E. Hernández-Orallo, J. Herrera-Tapia, K. Kuladinithi, V. Kuppusamy, P. Manzoni, A. Bin Muslim, A. Udugama, and Z. Vatandas, "Simulating Opportunistic Networks: Survey and Future Directions," *IEEE Communications Surveys and Tutorials*, vol. 20, no. 2, 2018.

[17] J. Herrera-tapia, P. Manzoni, C. T. Calafate, and J.-c. Cano, "Selecting the Optimal Buffer Management for Opportunistic Networks both in Pedestrian and Vehicular Contexts," *14th IEEE Annual Consumer Communications Networking Conference (CCNC 2017)*, pp. 395–400, 2017.

[18] E. Hernández-orallo, J. Herrera-tapia, J.-c. Cano, C. T. Calafate, and P. Manzoni, "Evaluating the Impact of Data Transfer Time in Contact-Based Messaging Applications," *IEEE Communications Letters*, vol. 19, pp. 1814–1817, 2015.

[19] Open Garden, "FireChat – Open Garden,"

[20] E. Hernández-orallo, D. Fernández, J. Herrera-tapia, J.-c. Cano, C. T. Calafate, and P. Manzoni, "GRChat : A Contact-based Messaging Application for the Evaluation of Information Diffusion," no. c, pp. 32–33, 2016.

[21] C. Boldrini, M. Conti, and A. Passarella, "Modelling Data Dissemination in Oppor tunistic Networks," *Proceedings of the third ACM workshop on Challenged networks*, pp. 89–96, 2008.

Gestión de la seguridad de las comunicaciones para entornos de HPC en centros de supercomputación

David Cortés-Polo, Felipe Lemus-Prieto, Jesús Calle-Cancho, Luis Ignacio Jiménez y José-Luis González-Sánchez [1]

Resumen— **La búsqueda de la eficiencia por parte de los centros de supercomputación para adaptarse a las necesidades de los usuarios, en cuanto a los recursos de cómputo, está llevando a integrar múltiples tecnologías que deben convivir de forma heterogénea, de manera que la gestión de los recursos, así como, la seguridad de la infraestructura de cómputo se está volviendo más compleja que en los escenarios tradicionales. En este marco, tecnologías como las redes definidas por software o la virtualización de funciones de red están siendo adoptadas como mecanismos que facilitan la gestión de las comunicaciones, y como soluciones que introducen un buen número de herramientas a la hora de gestionar la seguridad de los clústers de computación. En este trabajo se describen las principales características de la evolución de estas redes de comunicaciones y su aplicación a la seguridad de la información y las comunicaciones de un centro de cómputo, de forma que se expone un ejemplo de uso donde se aplican estas tecnologías de nueva generación para detectar y mitigar un problema de seguridad mediante funciones de red virtualizadas desplegadas en contenedores.**

Palabras clave— **Supercomputación, Virtualización, Comunicaciones, NFV, SDN, Seguridad**

I. Introducción

El panorama de la computación científica ha cambiando de forma muy rápida en los últimos diez años. La aparición de tecnologías como la virtualización de los recursos de cómputo han producido cambios significativos en la forma de planificar y configurar los mismos, e incluso de cómo interaccionan recursos de cómputo locales con otros distribuidos.

Actualmente, los centros de supercomputación buscan ofrecer recursos de computación escalables, muchas veces asociados a una institución o grupo, adaptándose a las necesidades concretas de cada tipo de usuario. De hecho, esta forma de particionar los recursos busca la reproducibilidad de los experimentos en múltiples infraestructuras de computación, para que cualquier investigador pueda, a partir de esos experimentos, continuar con el trabajo previamente realizado.

Por este motivo, los entornos de software modulares [1], entornos virtuales [2] o software científico desplegado en contenedores [3] se han convertido en las herramientas de uso diario en los centros de supercomputación, en los cuales los investigadores desarrollan sus aplicaciones y éstas son lanzadas sobre los recursos de cómputo implementando, además, garantías de reproducibilidad en los experimentos. Por contra, esta virtualización de recursos en un supercomputador hace mucho más compleja la gestión de la seguridad de la infraestructura. Anteriormente, mecanismos como los cortafuegos, software o hardware, o las listas de control de acceso eran suficientes para gestionar de forma eficiente el acceso a la infraestructura y el uso que se le daban a los recursos. Diferentes motivos como el incremento de servicios virtualizados, la interconexión de máquinas en remoto, o el uso de recursos en plataformas *cloud* para incorporarlos al cómputo local, han hecho que los mecanismos de seguridad tradicionales no sean suficientes para asegurar que los recursos de cómputo estén siendo usados de forma correcta y equitativa con respecto al resto de los usuarios.

Por todo esto, los protocolos de comunicaciones, así como el hardware de red que se está empezando a desplegar en los centros de supercomputación, no sólo atienden a los antiguos estándares y formas de computación, sino que se están migrando a las nuevas tecnologías definidas por software como las redes definidas por software (Software Defined Networks, SDN) o la virtualización de funciones de red (Network Function Virtualization, NFV) como dos de las principales tecnologías.

Estas dos tendencias en las comunicaciones, basadas en software, proveen una mayor versatilidad a las redes de comunicaciones de forma que pueden ser fácilmente integrables en el nuevo paradigma de computación que está originándose en los centros de supercomputación, siendo, además, herramientas indispensables para gestionar la seguridad en este nuevo paradigma [4].

Por tanto, las arquitecturas SDN y NFV aportan un buen número de beneficios a este nuevo paradigma de arquitecturas de computación y, por tanto, la integración de ambas en un único marco denominado red definida por software con funciones virtualizadas (Software Defined NFV, o SDNFV) [5]. De hecho, éste es un tema muy activo en la investigación y la industria, debido al control y la capacidad heredada de las redes SDN y la flexibilidad obtenida por la virtualización de las funciones proporcionadas por NFV. Es por esto que a través de una SDNFV, se pueden incluir mejoras importantes en los mecanismos de detección

[1]Centro Extremeño de Investigación, Innovación Tecnológica y Supercomputación (CénitS), e-mail: david.cortes@cenits.es.

y mitigación de los problemas de seguridad en la red.

El resto del artículo se organiza de la siguiente manera; la sección 2 presenta las redes definidas por software y la virtualización de funciones de red, así como la relación existente entre ambos conceptos. La sección 3 introduce la arquitectura de red SDNFV usada en este trabajo, así como el diseño y el desarrollo de los protocolos que la componen. En la sección 4, se presenta la detección y la mitigación de amenazas de seguridad en una red basada en SDNFV, describiendo el algoritmo para detectar un posible problema de seguridad así cómo los mecanismos para mitigarlo usando funciones de red virtualizadas en un escenario real. Finalmente, la sección 5 contiene las conclusiones finales del trabajo.

II. Nuevas tendencias en las redes de comunicaciones: SDN y NFV

La tecnología SDN surge como un nuevo paradigma de red, cuya principal característica es la separación del plano de datos del plano de control, con el objetivo de simplificar la gestión y configuración de las redes tradicionales [6]. Este tipo de redes proporcionan una vista global de la red a través de un controlador, por lo tanto, el plano de control queda centralizado en el controlador de la red que, a su vez, gestiona el plano de datos a través de protocolos abiertos como es el protocolo OpenFlow.

La Fig. 1 describe la arquitectura básica de SDN desde el punto de vista lógico. Como se pude observar, la arquitectura de SDN se divide en tres capas: aplicación, control e infraestructura. El plano de control y el plano de datos son las diferentes capas resultantes de la separación de las funciones de control y reenvío, de forma que la arquitectura proporciona más información a las aplicaciones sobre el estado de la red que los protocolos usados en las redes tradicionales, gracias a la presencia del controlador de red propuesto por SDN.

Esta tecnología proporciona agilidad, permitiendo una gestión dinámica de flujos y optimizando recursos ante las necesidades cambiantes de las aplicaciones que corren los usuarios en el clúster, que pueden tener diferentes requerimientos en cuanto a características y calidad de servicio [7].

Por otro lado, NFV aporta grandes ventajas en el aprovisionamiento de servicios en las redes de nueva generación. Este paradigma tiene como principal objetivo el desacoplar las funciones de red de los dispositivos físicos en los cuales se ejecutan. Además, NFV tiene el potencial de facilitar y flexibilizar el despliegue de nuevos servicios con más agilidad [8], permitiendo alcanzar los requisitos de baja latencia y alta fiabilidad requerida por las aplicaciones que se van a ejecutar en los clústers de cómputo [9].

Tal y como se muestra en la Fig. 2, los paradigmas SDN y NFV están íntimamente relacionados [10] y, con una integración eficiente de ambos, se podría conseguir un importante ahorro de costes y una mayor flexibilidad en la provisión de servicios. A la integración de ambos paradigmas en un único entorno se conoce como una red SDNFV.

Finalmente, aunque ambas soluciones son complementarias, no son dependientes la una de la otra, de forma que pueden ser implementadas por separado, siendo posible implementar SDN, NFV o una combinación de ambas. De acuerdo con el *White Paper* de NFV [11]: El objetivo de la virtualización de las funciones de red es que se puedan usar sin aplicar mecanismos de SDN, de forma que se basen en las técnicas que actualmente hay en la mayor parte de los centros de datos. A pesar de esto, las aproximaciones que se basan en la separación del plano de control y de reenvío como la propuesta por SDN mejora el rendimiento, simplifica la compatibilidad con despliegues existentes y facilita los protocolos de gestión y mantenimiento.

III. Implementación de una red basada en SDNFV para desplegar mecanismos de seguridad en un centro de supercomputación

Una red basada en SDNFV define servicios de red complejos que se ejecutarán en un hardware de propósito general, reemplazando el tradicional hardware específico creado para realizar las funciones

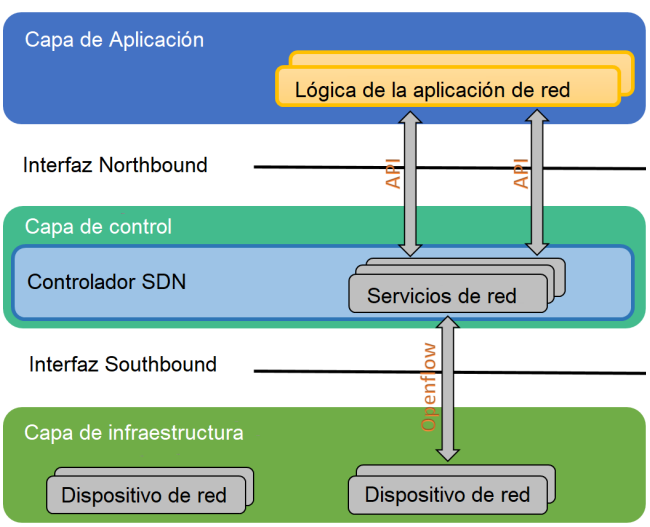


Fig. 1. Arquitectura de SDN

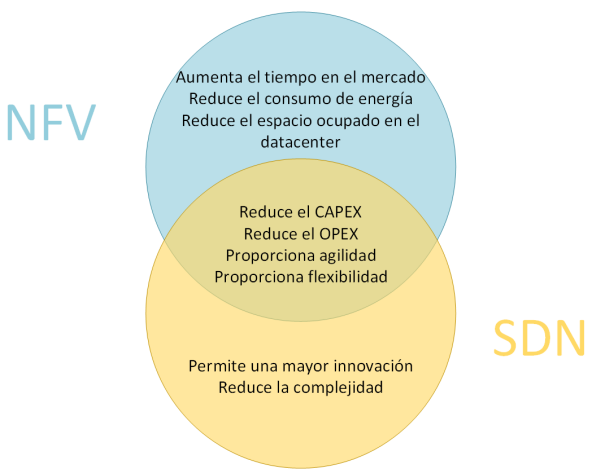


Fig. 2. Relación entre SDN y NFV

de comunicación [8].

En una arquitectura SDN pura, el controlador tiene la función principal de gestionar el plano de control y cómo deben conmutarse los flujos en la red. Esta función se mantiene en una red basada en SDNFV, pero debe complementarse con otra funcionalidad principal, que es la orquestación de recursos virtualizados en la red.

Esta orquestación se usa para implementar las funciones virtualizadas y administrar el ciclo de vida de la función. Desde un punto de vista de seguridad, la función del controlador/orquestador es muy importante porque gestiona servicios clave cuyo propósito es detectar y mitigar un ataque, reservar recursos de red para crear *black holes* o analizar los flujos de datos de la comunicación.

La Fig. 3, muestra la arquitectura y cómo ambos paradigmas se interconectan mediante el controlador/orquestador desplegado en la red. Como se puede observar en la figura, la arquitectura se divide en diferentes tipos de nodos que están interconectados en la red. Hay dos tipos de *switches* SDN en la arquitectura de red SDNFV propuesta.

El primero de ellos está desarrollado sobre CPqD/ofsoftswitch13, el cual es una versión de OpenFlow 1.3 Software Switch [12] ejecutado en espacio de usuario y que implementa todas las funcionalidades del estándar. Este switch ha sido modificado por el grupo de trabajo del proyecto BEhavioural BAsed forwarding (BEBA) para introducir un conjunto extendido de acciones y primitivas diseñadas para monitorizar el tráfico de la red y mejorar la seguridad de las comunicaciones [13]. El enfoque BEBA es una implementación de código abierto de un controlador y un conmutador OpenState, que se puede encontrar en [14].

El otro tipo de switch incluido en el escenario es Open vSwitch (OVS), que es un módulo del kernel que admite OpenFlow y reemplaza la implementación del *bridge* de Linux [15]. Ambos *switches* están gestionados por un controlador Ryu OpenFlow [16].

Ryu administra el plano de control de los diferentes tipos de conmutadores para habilitar el enrutamiento inteligente en la red SDNFV. Este controlador también implementa funcionalidades de orquestador para administrar los recursos de la infraestructura de virtualización de funciones de red (NFVI).

En esta arquitectura, la infraestructura para implementar las funciones de red virtualizadas es un clúster de contenedores basados en tecnología *docker*. Estos contenedores *docker* están integrados en la infraestructura como nodos de la red, a los que se puede llegar por diferentes rutas. Todas las funcionalidades virtualizadas se implementan en esos servidores, y el controlador debe modificar las rutas para incluir las funcionalidades NFV en la red e interactuar con los flujos.

En las siguientes subsecciones se describen en mayor detalle las tecnologías usadas en la red SDNFV por separado, especificando la implementación de SDN y de NFV usadas para, a través de la conjunción de ambas, desarrollar un sistema de detección y mitigación de un ataque real.

A. Implementación de la red SDN

En esta arquitectura la red SDN está compuesta por dos tipos de conmutadores. Los conmutadores OVS no son conmutadores inteligentes y su función principal es conmutar por la red los paquetes siguiendo las reglas dictadas por el controlador. Este conmutador también se implementa en los diferentes clústers de contenedores usados para desplegar los *dockers* que implementan la arquitectura NFV.

El conmutador CPqD con soporte de BEBA implementa la propuesta OpenState [17], ampliando las funcionalidades principales de OpenFlow e incluyendo las capacidades para aplicar diferentes reglas basadas en la coincidencia con diferentes estados descritos en las tablas de flujo SDN del conmutador.

Con esta funcionalidad, el conmutador adquiere la capacidad de reaccionar ante eventos a nivel de paquete, por ejemplo, analizando los flujos que se están conmutado. Si el resultado del análisis coincide con las reglas que el conmutador tiene en las tablas de flujo, entonces puede actuar de acuerdo con la regla. Esto se debe a que OpenState es una extensión de OpenFlow, y utiliza una tabla de flujo, que busca coincidencia dentro de la misma ejecutando cierta acción. La extensión agrega una tabla de estados adicional que contiene los estados de flujo.

El conmutador usado en este trabajo primero comprueba el paquete con el estado en el que está la ejecución para, posteriormente, realizar la acción contenida en la tabla de flujo. La Fig. 4 muestra la arquitectura de OpenState. Gracias a esta arquitectura se ha conseguido que la implementación de las aplicaciones de red se simplifique y aumente el rendimiento de las mismas [17].

Se ha modificado el código del *switch* CPqD/ofsoftswitch13 para implementar las capacidades OpenState. Para ello se han introducido como una extensión experimental de OpenFlow. Para poder realizar esto, la especificación OpenFlow define la estructura común de los campos experimentales, con un conjunto de mensajes y acciones que deben ser definidos. Cada desarrollador puede personalizar el formato de cada estructura, siendo el campo *experimenter_id* el identificador adoptado para definir la extensión que se está implementando. En nuestro caso, la extensión se ha definido como *0xBEBABEBA*, debido a que se está haciendo uso del código desarrollado por el proyecto BEBA.

B. Implementación de las funciones NFV

La arquitectura NFV se implementa mediante *dockers* en un clúster de contenedores. El controlador de red debe conocer la información del clúster la cual incluye información del hardware implementado así como el rango de IP usado por

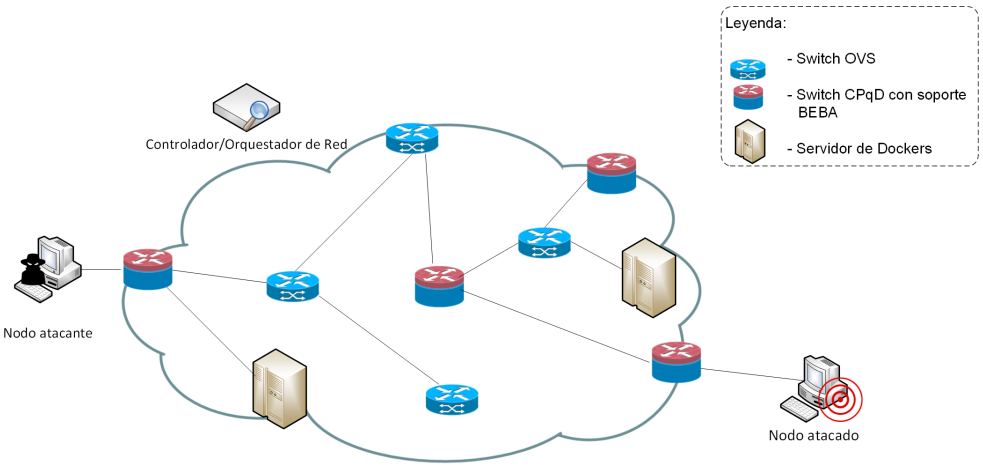


Fig. 3. Red SDNFV propuesta

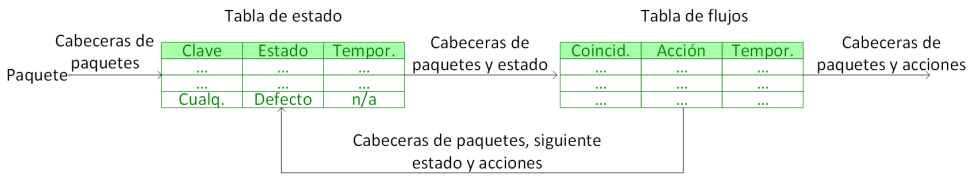


Fig. 4. Arquitectura OpenState

el cluster. De forma que todas estas características definen el NFVI.

Con esta información, el controlador puede gestionar los recursos y desplegar las funciones virtualizadas. Como ya se ha descrito anteriormente, en este ecosistema, las funciones de la red están empaquetadas en contenedores *docker*. Con este enfoque, la implementación de la función es independiente de la plataforma utilizada para desplegarla.

La Fig. 5 muestra la arquitectura del NFVI y cómo se virtualizan las funciones. Como se puede observar en la figura, NVFI se interconecta con la red SDNFV a través de interfaces de red física (pNIC). Estas NIC intercambian los paquetes entre las funciones de red virtualizadas y la red SDN. Cada función virtualizada incluida en un contenedor tiene dos interfaces de red virtual, que están interconectadas por medio de un puente [18].

Con este enfoque, un pequeño clúster puede

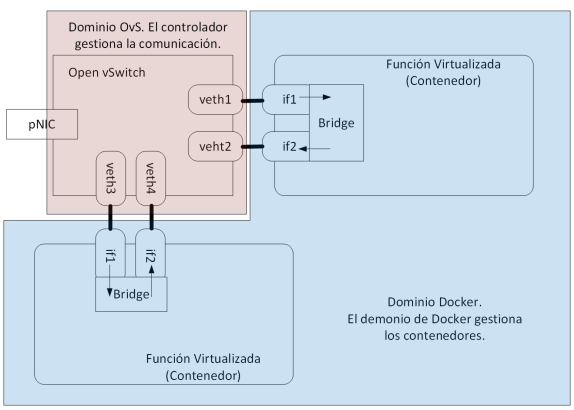


Fig. 5. Arquitectura NFVI

implementar un gran número de instancias, configurando la conexión de red y las interfaces que permiten la comunicación entre los contenedores. Los *dockers* implementan su funcionalidad básica en un fichero conocido como *Dockerfile*. Este archivo incluye un conjunto de instrucciones para construir automáticamente el entorno en una imagen de *docker*. Esta imagen crea una instancia en el NFVI para ejecutar la función en la red SDNFV. La comunicación entre el controlador/orquestador y el NFVI se cifra mediante un canal seguro para enviar la configuración de la función de red virtualizada, crear el *Dockerfile* y los comandos necesarios para implementar la función virtualizada en la infraestructura.

C. Implementación del Controlador/Orquestador

El controlador desplegado en la arquitectura es Ryu. Este controlador es soportado por los laboratorios NTT y está basado en componentes que son desarrollados para implementar las distintas funcionalidades.

Existen un número considerable de componentes básicos incluidos en el controlador, siendo éstos modificados y ampliados para aumentar las funcionalidades de aplicación que gestiona el controlador, adaptándose a los requisitos del problema.

La Fig. 6 describe la arquitectura de Ryu y cómo las aplicaciones SDN interactúan con los conmutadores OpenFlow.

Para incluir las funciones básicas de BEBA en Ryu, se deben implementar nuevos mensajes, acciones y campos de comprobación. De hecho, para que el controlador gestione las capacidades BEBA,

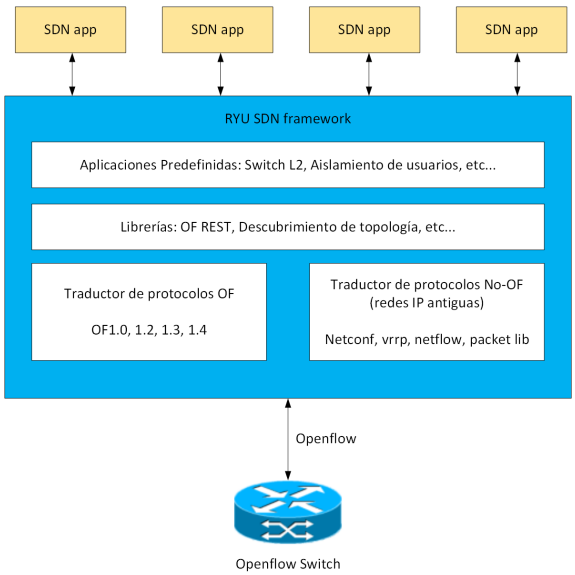


Fig. 6. Arquitectura Ryu

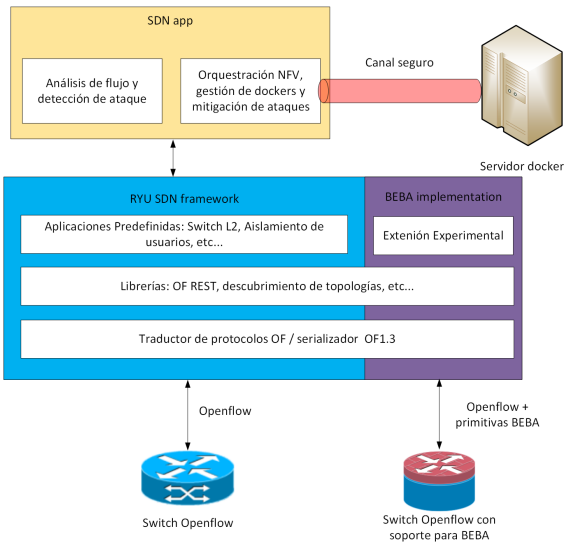


Fig. 7. Arquitectura controlador/orquestador

debe ser extendido desde una implementación básica, incluyendo las capacidades de gestión de toda la información que proporcionan las diferentes tablas de gestión de flujo.

La implementación de BEBA utiliza la funcionalidad aportada por Openflow para incorporar mensajes experimentales, usados para implementar la lógica de control, así como definir el *payload* de los paquetes que deben transportar la información requerida para conmutar el tráfico, tomar decisiones sobre los flujos o ejecutar reglas en la red.

Para este trabajo, todas las modificaciones se han realizado sobre los *switches* CPqD, de forma que, a la instalación básica de Ryu, se le ha incorporado la implementación de BEBA. Esta implementación ha sido extendida en su funcionalidad para incluir el orquestador de NFV, obtener la información del NFVI y desplegar las funciones virtuales sobre un el clúster de *docker*, implementando la funcionalidad de corta fuegos y ser capaz de configurar la red de forma que redirija el tráfico a la función virtualizada de red.

La Fig. 7 describe la arquitectura completa del controlador/orquestador. Como se puede observar, la arquitectura de Ryu se amplía para integrar las primitivas de BEBA, de forma que la aplicación SDN puede hacer uso de esas primitivas para desarrollar nuevas funcionalidades como la que aquí se presenta.

Un aspecto clave es que la red SDNFV es heterogénea. El controlador debe gestionar ambos conmutadores e implementar un protocolo de conmutación integrado para enrutar los flujos en la red. Un ejemplo de esa integración de dispositivos y comportamientos heterogéneos es la función de descubrimiento de MACs de los nodos. Los *switches* OpenFlow con soporte BEBA implementan un protocolo de descubrimiento y aprendizaje de MAC para enrutar los flujos [19], mientras que los conmutadores OpenFlow sin capacidades especiales se gestionan mediante un protocolo de conmutación simple que descubre la dirección MAC de cada nodo la primera vez que ha de enviarle un flujo de datos.

La aplicación desarrollada para este trabajo se sustenta sobre BEBA para analizar los flujos y detectar los que son sospechosos. Con esta información, el módulo de orquestación NFV elige el mejor servidor de *dockers* para desplegar el contenedor que implementa la función NFV. Para ello se crea un canal seguro y se ejecutan los comandos necesarios para su despliegue. Una vez se ha realizado la implementación, el controlador modifica las tablas de flujo para redirigir el tráfico seleccionado al contenedor NFV que mitigue el ataque.

IV. Detección y mitigación de un ataque usando una red SDNFV.

Como se describe en la sección anterior, el controlador/orquestador implementa dos módulos para detectar y mitigar los ataques. La fase de mitigación requiere la información del flujo; IP de origen, puerto de origen, IP de destino, puerto de destino y el protocolo utilizado. Toda esta información se obtiene en la fase de detección. El módulo de detección es un subproceso que, periódicamente, envía una solicitud a los conmutadores OpenFlow para adquirir información estadística. El bucle principal del análisis de los flujos es el siguiente:

Algoritmo 1 Análisis de flujos

```
1: mientras verdad hacer
2:     Envia peticiones de estados a la tabla de estado.
3:     para datapath en datapaths hacer
4:         Obtener estadisticas del datapath
5:         Espera X segundos
6:     fin para
7: fin mientras
```

En este bucle se solicitan las estadísticas de los flujos al conmutador, utilizando la primitiva *OFPExpStateStatsMultipartRequestAndDelete* implementada en BEBA. Para recibir una respuesta del conmutador, se ha creado un controlador de eventos que recibe el mensaje *EventOFPExperimenterStatsReply* con la obtención de las estadísticas.

Estas estadísticas se analizan obteniendo la IP origen y destino del paquete, así como el puerto destino y el tipo de protocolo de nivel de transporte, TCP o UDP, para recontar los diferentes flujos que se están analizando.

Una vez que se analizan los flujos, se calcula la entropía. Ésta se usa para detectar ataques DDoS midiendo las propiedades estadísticas del encabezado del paquete. En este caso, la muestra de datos reales analizados, se basa en la comparación de los paquetes consecutivos de un flujo para identificar un ataque, de forma que la entropía puede ser tratada como se muestra en la Ecuación 1:

$$E = -\sum_{i=1}^{n} p_i log_2 p_i \quad (1)$$

Donde E es la entropía, n el número de elementos detectados en el análisis de los flujos y p_i la probabilidad de encontrar el elemento *iésimo* en la conjunción de elementos detectados en el análisis. En este trabajo, hemos utilizado la IP de origen, de destino, el puerto de origen y el puerto de destino como elementos para detectar la entropía. Una vez calculada la entropía, se ejecuta el algoritmo de detección. El algoritmo se basa en las Ecuaciones 2 a 4.

$$linf = \overline{x_p} - precision * \sigma_p \quad (2)$$

$$lsup = \overline{x_p} + precision * \sigma_p \quad (3)$$

$$Ataque = \begin{cases} falso & Si \quad linf < x < lsup \\ verdadero & cualquier \quad otro \end{cases} \quad (4)$$

Donde $\overline{x_p}$ es el valor medio de los elementos analizados y la *precisión* es el valor para definir la precisión en el algoritmo de detección. En éste, los valores utilizados para la precisión son de un 68% para precisión pequeña, de un 95% para precisión media y un 99.7% para una precisión alta. Por último, σ_p es la desviación estándar.

Si se detecta el ataque, el controlador/orquestador busca en una base de datos donde se encuentra la información sobre el clúster de *dockers* para elegir el servidor que puede implementar de forma eficiente la función NFV. Una vez que se elige el servidor, el controlador abre un canal seguro para desplegar el NFV en un contenedor *docker*. El despliegue se realiza mediante el archivo descriptor de *Dockerfile*. Este archivo describe la función NFV y el comportamiento del *docker*. El código 2 muestra un ejemplo de *Dockerfile*.

Algoritmo 2 Ejemplo fichero Dockerfile

```
# Firewall que permite trafico
# del puerto 80
FROM base
ENTRYPOINT ifinit && \
brinit && \
iptables -A FORWARD -p tcp && \
–dport 80 -j ACCEPT && \
iptables -A FORWARD -j DROP && \
/bin/bash
```

El *Dockerfile* define las reglas con las que se lanzará el *docker*. Una vez que se construye el contenedor *docker*, se crean las interfaces y se establece el *bridge* entre ellas. Todo ello es desplegado en el servidor elegido entre todos los del clúster de *dockers*. El controlador, una vez finalizada la fase de despliegue, modifica las tablas de reenvío de los conmutadores involucrados en la comunicación del ataque DDoS para mitigarlo. Con este mecanismo, los conmutadores que reciben el ataque solo tienen que reenviar los paquetes al puerto de salida, y el ataque se mitiga utilizando un servidor diseñado para absorverlo sin que se produzca un DDoS en el nodo destino elegido por el atacante. En este caso, el destino es el firewall desplegado como una función NFV, el cual es una implementación software de firewall con las reglas pertinentes para filtrar el tráfico malicioso.

V. Conclusiones

Debido a la evolución en la forma de ejecuta los trabajos de cómputo en los centros de supercomputación, así como en las nuevas tecnologías que se van adaptando para la reproducibilidad de los trabajos, se hace necesario una adaptación de los mecanismos tradicionales en los que se basan los centros de computación para gestionar los recursos de cómputo. Este trabajo presenta una novedosa arquitectura SDNFV para ser aplicada en una red de computación que admite la detección de ataques DDoS y mitiga el ataque utilizando funciones virtualizadas en un contenedor de aplicaciones. La arquitectura presentada es una red simple y rentable basada en SDN que analiza los flujos y detecta los ataques utilizando un mecanismo de entropía desarrollado en el controlador. Este mecanismo es más avanzado en términos de detección y mitigación, porque solo se mitiga el flujo involucrado en el ataque. De hecho, el mecanismo puede discriminar diferentes valores como IP, puertos o protocolos. Se pueden implementar diferentes funciones virtualizadas según la aplicación que se ejecute en la parte superior del controlador Ryu. En este artículo presentamos la función virtualizada de un cortafuegos, pero las

capacidades de la arquitectura desarrollada permiten implementar otras funciones como el conformado del tráfico, la inspección profunda de paquetes, analizadores de flujo de datos, etc. La simplicidad de la arquitectura desarrollada hace transparente el despliegue de las funcionalidades de la red para el usuario final. El controlador/orquestador gestiona el plano de control y administra los recursos de la red para aumentar el rendimiento.

Agradecimientos

Este trabajo está financiado en parte por los Fondos Europeos de Desarrollo Regional de Extremadura a través de la Fundación Computación y Tecnologías Avanzadas de Extremadura (COMPUTAEX), mediante el proyecto GR18195.

Referencias

[1] Furlani JL, Osel PW. Abstract Yourself With Modules In *Proceedings of the 10th USENIX Conference on System Administration. LISA'96*, 1996.

[2] Smith JE, Nair R. Virtual Machines: Versatile Platforms for Systems and Processes. In *The Morgan Kaufmann Series in Computer Architecture and Design Series. Morgan Kaufmann Publishers*, 2005.

[3] Kurtzer GM, Sochat V, BauerMW. OpenFlow Experimenter Labels for Encoding Adaptive Network Functions In *arnoff Symposium 2018 IEEE 39th* , pp. 1-5, 2018.

[4] Higgins, J., Holmes, V., and Venters, C. Securing user defined containersfor scientific computing. In *High Performance Computing & Simulation (HPCS),2016 International Conference on*,2016.

[5] W. Wang, Y. Liu, Y. Li, H. Song, Y. Wang and J. Yuan. Consistent State Updates for Virtualized Network Function Migration. In *IEEE Transactions on Services Computing.*, 2017.

[6] B. A. A. Nunes, M. Mendonca, X.-N. Nguyen, K. Obraczka, and T. Turletti. A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks. In *IEEE Communications Surveys & Tutorials*, vol. 16, no. 3, pp. 1617-1634, 2014.

[7] W. Chin, Z. Fan, and R. Haines. Emerging technologies and research challenges for 5G wireless networks. In *IEEE Wireless Communications*, vol. 21, no. 2, pp. 106-112, 2014.

[8] B. Han, V. Gopalakrishnan, L. Ji, and S. Lee. Network function virtualization: Challenges and opportunities for innovations In *IEEE Communications Magazine*, vol. 53, no. 2, pp. 90-97, 2015.

[9] Hayashida, Mami and Rivera, Sergio and Griffioen, James and Fei, Zongming and Song, Yongwook. Debugging SDN in HPC Environments In *Proceedings of the Practice and Experience on Advanced Research Computing*, 2018.

[10] Mijumbi, R., Serrat, J., Gorricho, J. L., Bouten, N., De Turck, F., and Boutaba, R. Network function virtualization: State-of-the-art and research challenges In *IEEE Communications Surveys & Tutorials*, vol 18(1), 236-262. 2015.

[11] ETSI NFVISG. Network functions virtualization, white paper In *https://www.etsi.org/technologies/nfv*, Último acceso 2019-06-04.

[12] CPqD OpenFlow 1.3 Software Switch In *http://github.com/CPqD/ofsoftswitch13*, Último acceso 2019-06-04.

[13] Bifulco, R., and Matsiuk, A. Towards scalable sdn switches: Enabling faster flow table entries installation In *ACM SIGCOMM Computer Communication Re-view*, vol 45(4), 343-344. 2015.

[14] BEBA BEBA SDN project home page In *http://www.beba-project.eu*, Último acceso 2019-06-04.

[15] Pfaff, B., Pettit, J., Koponen, T., Jackson, E., Zhou, A., Rajahalme, J., Am-idon, K. The design and implementation of open vswitch. In *2th USENIX sympo-sium on networked systems design and implementation (NSDI 15)*, 2015.

[16] RYU project team RYU SDN Framework In *https://osrg.github.io/ryu/re-sources.html#books*, Último acceso 2019-06-04.

[17] G. Bianchi, M. Bonola, A. Capone, and C. Cascone OpenState: program-ming platform-independent stateful OpenFlow applications inside the switch. In *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 2, pp. 44?51, Apr. 2014.

[18] Nakagawa, Y., Lee, C., Hyoudou, K., Kobayashi, S., Shiraki, O., Tanaka, J., and Ishihara, T. Dynamic virtual network configuration between containers using physical switch functions for NFV infrastructure. Network Function Virtualization and Software Defined Network (NFV-SDN) In *IEEE Conference on. IEEE*, 2015.

[19] Pontarelli, S., Bonola, M., Bianchi, G., Capone, A., and Cascone, C. Stateful openflow: Hardware proof of concept. In *IEEE 16th International Conference on High Performance Switching and Routing (HPSR)* , 2015.

Aplicación del coeficiente de correlación de Pearson en Cloud Computing para la optimización de CPU y ancho de banda

Sergi Vila Almenara[1] , Fernando Guirado[2] , Josep L. Lérida[3] y Fábio Verdi[4]

Resumen— **Los entornos Cloud permiten satisfacer de forma dinámica mediante virtualización las necesidades de recursos de computación del usuario. El uso dinámico de recursos condiciona la demanda de hosts en funcionamiento. Mediante migraciones de máquinas virtuales, los datacenters realizan balanceos de carga para optimizar el uso de recursos y resolver saturaciones. En este trabajo se ha implementado una heurística para elegir cuales son las máquinas más adecuadas para ser migradas. Esta heurística se basa en el coeficiente de correlación de Pearson teniendo en consideración la carga de CPU y del ancho de banda. Utilizando el simulador CloudSim, se ha comparado la heurística propuesta con el resto de técnicas de la literatura, alcanzando una reducción media del 53% del número de migraciones con ligeras mejoras en el uso del ancho de banda (2.5%), saturación de la red (5.7%), energía (5%) y hosts utilizados (6.9%).**

Palabras clave— **Cloud computing, planificación, máquinas virtuales , migraciones, correlación Pearson, balanceo de carga, CloudSim.**

I. Introducción

El auge de Internet y, recientemente, el uso de Big Data ha llevado a las empresas a extender sus servicios en datacenters, ofreciendo una forma fácil y escalable de satisfacer las necesidades de recursos sin requerir de una inversión ingente en tecnología e infraestructura.

El objetivo de los proveedores de estos servicios es rentabilizar el uso de sus máquinas con el fin de incrementar los beneficios. Esto se consigue maximizando la ocupación de los recursos disponibles. Esta optimización posibilitará una reducción en el consumo eléctrico, extender su vida útil, disponer de nuevos recursos en caso de fallo, etc. Sin embargo, deberá existir un balanceo entre el uso de las máquinas y su desempeño, teniendo que cumplir el contrato acordado con sus clientes (Service Level Agreement), garantizando cierta cantidad de recursos en un tiempo establecido.

Dependiendo de las necesidades de los clientes en cada momento, la demanda de recursos será variable, mediante migraciones de máquinas virtuales (MVs) realizaremos balanceos de carga, transfiriendo las MVs a hosts más adecuados. Si se alcanza una demanda mayor a la esperada, nuevas máquinas físicas deberán ser desplegadas, migrando las máquinas virtuales que sean necesarias a estos nuevos hosts para garantizar los recursos solicitados. En el caso en que el desempeño de las MVs sea menor, las MVs de varios hosts podrán ser agrupadas, permitiendo apagar aquellos que no se utilizan, con el consecuente ahorro de energía.

Los retos a los que se enfrenta el balanceo de carga en entornos Cloud son: detección de sobre/infrautilización de los hosts, selección de máquinas virtuales a migrar, selección de hosts destinatarios (incluyendo qué procesadores), selección de hosts a activar/desactivar, gestión de las comunicaciones entre MVs, etc.

Ante esta gran cantidad de desafíos, se deben estudiar cada uno de ellos teniendo en cuenta el posible impacto negativo en el resto. En el presente trabajo, se propone una heurística para decidir qué máquinas virtuales tienen un mayor impacto en el uso de CPU y ancho de banda (bandwidth, BW) del host saturado. Esta heurística se basa en el coeficiente de correlación de Pearson teniendo en consideración tanto el uso de CPU como de ancho de banda utilizados. Para llevar a cabo este trabajo ha sido necesario desarrollar un modulo en CloudSim que permita modelizar y simular el comportamiento de la red.

El resto del trabajo se organiza como sigue: en la Sección II se describe el estado del arte en el que se ha basado el presente trabajo, en la Sección III, se presenta el método de correlación de Pearson, en la Sección IV se detallan la modificaciones realizadas en CloudSim, en la Sección V se describe los experimentos realizados y su configuración, en la Sección VI se muestran los resultados, y finalmente las conclusiones y el trabajo futuro se discuten en la Sección VII.

II. Estado del arte

Gracias al grupo de investigación de Buyya disponemos de CloudSim, el simulador utilizado en este trabajo, siendo su última versión compatible con contenedores [1]. Nuevas funcionalidades de CloudSim han sido incorporadas por proyectos como CloudSim Plus [2], MilpFlow [3], CloudAnalyst [4] y DartCSim+ [5]. Como alternativas a CloudSim también disponemos de iCanCloud [6], DockerSim [7] y Cloudlightning Simulator [8]. Aunque algunas de éstas ampliaciones incluyen la simulación de entornos de red, su uso está restringido al tipo de experimentos para los que han sido diseñadas, dificultando su

[1]INSPIRES, Universitat de Lleida, e-mail: svila@diei.udl.cat
[2]INSPIRES, Universitat de Lleida, e-mail: f.guirado@diei.udl.cat
[3]INSPIRES, Universitat de Lleida, e-mail: jlerida@diei.udl.cat
[4]Departamento de Computação (DComp), Universidade Federal de São Carlos Campus Sorocaba, e-mail: verdi@ufscar.br

adaptación para el propósito de éste trabajo.

En la literatura podemos encontrar distintos trabajos basados en la aplicación de correlaciones. Los trabajos de Douglas et al. [9], Chok et al. [10] Hauke et al. [11] and Winter et al. [12] nos permiten profundizar en el funcionamiento de las correlaciones de Pearson, Kendall y Spearman, comparándolas y analizando cómo la cantidad, calidad y continuidad de los datos alteran sus resultados.

Choudhary et al. [13] realizaron un experimento similar al presentado en este trabajo. Utilizando CloudSim, compararon las técnicas de selección de màquina virtual MC (Maximum Correlation) y RS (Random Selection) utilizando la técnica de asignación de host MAD (Median Absolute Deviation) con una técnica propia utilizando el coeficiente de Spearman teniendo en cuenta la CPU.

Moghaddam et al. [14] utiliza el coeficiente de correlación de Pearson para desaturar hosts, teniendo solo en cuenta la carga de CPU, a diferencia de nuestra propuesta donde también tiene en cuenta el ancho de banda.

Shaw et al. [15] se basa en un método de correlaciones opuestas de CPU, para decidir el mejor host para un grupo de MVs con el fin de equilibrar las variaciones de CPU.

Nilson et al. [16] propuso una heurística para migrar MVs de hosts saturados utilizando el coeficiente de Pearson en base al uso de CPU y los datos recibidos y enviados entre contenedores. Los experimentos en este trabajo se llevan a cabo en un entorno real reducido utilizando Docker y con cargas sintéticas. Partiendo de este trabajo, y trabajando en colaboración con los autores, se propone un modelo del uso de red que trate los canales de comunicación de las MVs de forma global, se realiza además una adaptación de CloudSim para considerar el uso de los canales de comunicación por parte de las MVs. De este modo llevamos la experimentación a un entorno CloudSim que nos permite modelar distintas topologias lógicas y físicas de connexión de los recursos, configurar distintas politicas de selección, asignación, etc. En definitiva, crear un entorno de simulación que nos permita realizar simulaciones mucho más complejas, adaptadas a la realidad.

III. Planteamiento del Problema

A. Correlaciones

La correlación entre dos conjuntos de datos ordenados es la relación lineal existente entre ellos, determinando su relación independientemente de su escala. El coeficiente de correlación de Pearson estima el nivel de dependencia entre dos cantidades. Asumiendo n muestras de dos variables x e y, el coeficiente de correlación de Pearson se calcula mediante la Eq. 1, donde $\bar{x}$ y $\bar{y}$ representan la media aritmética de x e y, respectivamente. Además, el orden de los elementos es determinante en el coeficiente obtenido.

$$r_{xy} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2 \sum_{i=1}^{n}(y_i - \bar{y})^2}} \quad (1)$$

TABLA I
Parámetros heurística

Símbolo	Definición
r_{cpu}	Coeficiente de correlación del uso de CPU
$r_{network}$	Coeficiente de correlación del uso de red
$weight$	Valor de ponderación, 0.5 por defecto
h_{val}	Valor heurístico obtenido mediante Eq. 2

La correlación entre dos conjuntos de elementos ordenados será directa cuando éstos sigan un patrón de variación similar, e inversa cuando sus valores evolucionen de manera opuesta. No habrá correlación cuando no se forme ninguno de los dos patrones descritos, siendo cada conjunto independiente del otro.

El rango de valores de la Eq. 1 es [-1, 1], siendo -1 una correlación inversa, 0 cuando no exista correlación, y 1, si ésta es directa.

B. Heurística

La heurística propuesta es una adaptación de la utilizada por Nilson et al. [16]. La Tabla I describe los parámetros utilizados para el cálculo de la heurística propuesta. En el presente trabajo modelamos el efecto de cada MV sobre el uso de la CPU y los canales de comunicación de forma global, sin distinguir entre canales TX y RX. Cuando el sistema detecta que un host está saturado, mediante la Eq. 1 calculamos los coeficientes de Pearson para CPU (r_{cpu}) y red ($r_{network}$) para cada MV en el host saturado. Una vez obtenidos los coeficientes de Pearson, calculamos para cada MV el valor de la heurística, h_{val}, mediante la Eq. 2.

$$h_{val} = \frac{1 - (weight * r_{cpu})}{(1 + (weight * r_{cpu}) - (weight * r_{network}))} \quad (2)$$

C. Ejemplo

Disponemos de un host con 4 MVs y detectamos que está sobrecargado, migrando una MV que utiliza mucha CPU solucionaríamos la sobrecarga de CPU. Observando en la Figura 1 comprovamos que las MVs 2 y 3 tienen un consumo más elevado de CPU (MIPS) y una mayor correlación con el host, serían buenas candidatas a ser migradas. Examinando el ancho de banda en la Figura 2, las MVs 0 y 1 son las que tienen una mayor correlación en relación al uso de los canales de comunicación del host. La Figura 3 muestra el valor h_{val} en función de los valores obtenidos por las correlaciones de las Figuras 1 y 2.

$$heurística = \begin{cases} migración, & \text{si } h_{val} <= 0.65 \\ nada, & \text{sino} \end{cases} \quad (3)$$

El valor h_{val} determina el coste-beneficio de realizar la migración de la MV. Debemos valorar hasta qué punto debemos reducir la carga de CPU del host sin que afecte significativamente a la red. A partir de la propia experimentación, se ha determinado que

este límite sea 0.65, si el valor h_{val} de una MV supera este valor, no debería ser migrada (Eq. 3). Por lo tanto, debemos fijarnos en los valores más pequeños que 0.65, siendo la MV 3 la candidata idónea para ser migrada ya que existe una correlación alta para la CPU y baja para el ancho de banda. En el caso de la MV 2, su correlación inversa para el ancho de banda favorece a compensar al resto de MVs, aunque seguiría siendo mejor opción que las MVs 0 y 1.

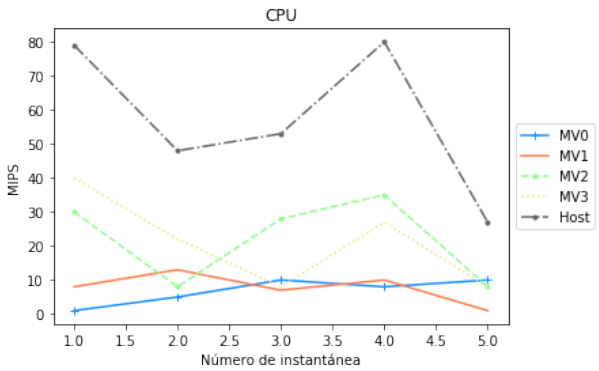


Fig. 1. Variación de las cargas de CPU

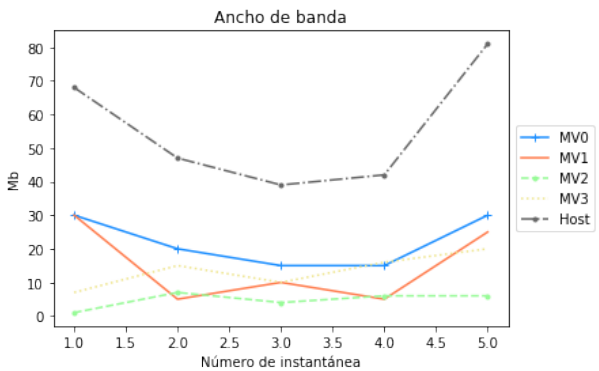


Fig. 2. Variación del ancho de banda

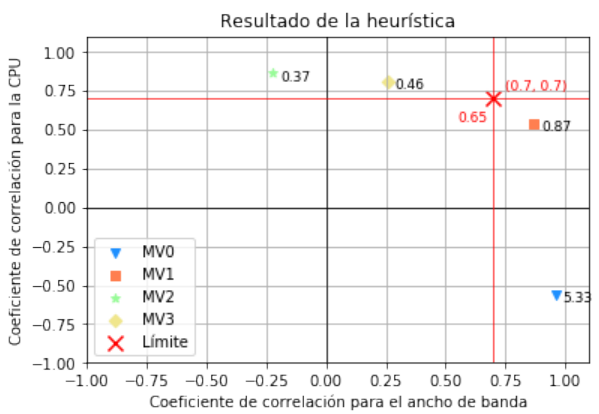


Fig. 3. Valores de la heurística

IV. CloudSim

CloudSim [17] permite la simulación de entornos Cloud. En los experimentos que permite CloudSim se dispone de datacenters, hosts, máquinas virtuales y contenedores [1]. Existen dos tipos principales de experimentación:

- Infraestructura predefinida donde se pretende simular la ejecución de un conjunto finito de tareas, con requerimientos fijos durante toda su ejecución y donde el objetivo principal es minimizar el tiempo de ejecución del conjunto de tareas.
- Ejecución de tareas indefinidas en el tiempo con un consumo de CPU variable, donde se persigue optimizar los recursos utilizados garantizando al mismo tiempo el cumplimiento del SLA.

En el presente trabajo nos vamos a centrar en el segundo tipo de experimentación, permitiendo realizar balanceo de carga de CPU y de red.

A. Implementación de Módulo de Red

Para poder validar el uso de correlaciones teniendo en cuenta el ancho de banda, se han ampliado las funcionalidades de CloudSim con el objetivo de permitir la generación de topologías, interacciones entre máquinas virtuales y la recopilación de instantáneas del estado del datacenter (almacenando los datos de MVs, hosts y red antes del proceso de migración). Así como nuevas métricas para conocer el estado de MVs, hosts y enlaces de red en cada momento.

A continuación se detallan las contribuciones realizadas:

- Creador de topologías del tipo estrella, árbol o una combinación de ambas, véase Figura 4.
- Generador de trazas sintéticas utilizando las técnicas cumSum y rollingWindow para obtener variaciones más suaves
- Interacciones con ancho de banda dinámico entre máquinas virtuales
- Creación de instantáneas, almacenando el estado de las MVs y los hosts
- Balanceo de flujos de datos mediante el algoritmo Dijkstra
- Nuevo sistema de exportación de métricas
- Capacidad para exportar toda la simulación en ficheros para su posterior análisis

B. Incorporación de la Heurística propuesta

La heurística ha sido implementada como una política de selección de máquina virtual del propio CloudSim ($PowerVMSelectionPolicy$). Redefiniendo la función $getVMToMigrate$ e implementando el Algoritmo 1. Una vez detectada la saturación de uno de los hosts, desplegamos la ejecución de nuestra heurística a través del Algoritmo 1. Este algoritmo se ejecutará tantas veces como CloudSim considere necesario hasta solucionar la saturación o hasta que no se seleccionen más máquinas virtuales para migrar. Primero, se obtienen las máquinas virtuales que no están siendo migradas (línea 2). En las líneas 3 y 4 se obtienen los coeficientes de correlación de Pearson para cada MV en relación al host utilizando una ventana de tiempo marcada por los últimos 12 capturas del estado de la CPU y red. En

el caso de la correlación de red, solo se tienen en cuenta las comunicaciones entre máquinas virtuales de ese mismo host. En la línea 5, obtenemos el resultado de la heurística para cada MV. Finalmente, en la línea 6 seleccionamos una MV cuyo valor h_{val} se acerque más a 0.65 sin sobrepasarlo, en el caso que no exista ninguna MV que cumpla este criterio, se devuelve valor nulo.

V. Configuración de la experimentación

En esta sección se describe la configuración del entorno de experimentación y las pruebas realizadas.

A. Trazas

Las trazas de carga de CPU utilizadas forman parte del entorno PlanetLab obtenidas con el sistema de monitoreo CoMon [18], un conjunto de trazas correspondientes a 10 días de ejecución con alrededor de 1000 máquinas virtuales. Se han utilizado los 100 primeros ficheros de la traza 20110303, conteniendo cada fichero 288 valores correspondientes a un día de ejecución.

Las trazas de red han sido generadas de forma sintética utilizando un generador propio[1]. Estas trazas están caracterizadas por ser fácilmente predecibles, sin alteraciones bruscas en sus valores, lo que permite obtener correlaciones directas o indirectas con más facilidad.

B. Máquinas virtuales

La Tabla II muestra la configuración de máquinas utilizadas. Cada máquina virtual contiene un cloudlet. Los cloudlets actúan como tareas sin fin, por ejemplo, un servicio web. Disponemos de 50 MVs pequeñas con 1000 MIPS y 50 de grandes con 2000 MIPS. El numero de MIPS requeridos para finalizar es ilimitado, siendo los porcentajes de carga de CPU de PlanetLab quien decide cuantos MIPS se ejecutan en cada instantánea.

C. Hosts

La Tabla III muestra las características de los 10 hosts utilizados. Estas características coinciden con las definidas por defecto por CloudSim. Se han utilizado 6 hosts pequeños con 1860 MIPS y 4 hosts grandes con 2660 MIPS, todos ellos con 2 CPUs. En la Tabla III se muestra el consumo de energía de los hosts.

D. Comunicaciones

Una interacción se define como la comunicación entre dos MVs durante toda la simulación. Asumimos para toda interacción un límite del uso del ancho de banda de 10 Mbps. A lo largo de la simulación, los valores de las interacciones serán actualizados utilizando las trazas de red.

La Tabla V muestra tres clases distintas de interacción definidas. La columna Proporción indica la probabilidad con que la clase aparece en las interacciones. La columna Rango define el rango de utilización del canal de comunicación para cada clase. Para cada interacción se define pues de forma aleatoria el tipo y rango según los valores de la tabla. Se ha establecido además que un 10% de las interacciones se produzcan dentro de un mismo host, mientras que solo un 0.05% se producen fuera del host.

E. Topología

Se toma como topología de partida para todos los experimentos la mostrada en la Figura 4. De los 10 hosts existentes, 6 hosts son utilizados para formar 6 grupos de MVs relacionadas entre sí, aunque también tienen interacciones con otras MVs, representando ésta topología lógica con líneas finas. El peso en las aristas que unen los nodos muestran la cantidad de Mb utilizados en el momento de realizar la captura de datos. Los switches están representados por los puntos centrales coloreados. La MVs en cada uno de los Hosts están representadas por los círculos discontinuos en cada uno de los clusters.

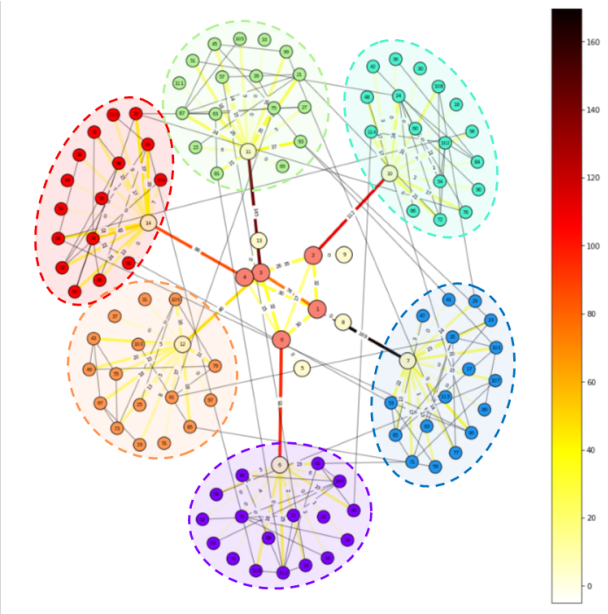


Fig. 4. Topología inicial

F. Parámetros

La Tabla VI muestra un resumen de los parámetros de configuración de CloudSim más importantes. Todos los experimentos han sido realizados utilizando estos parámetros. Se han realizado 30 experimentos para cada técnica variando el orden inicial de colocación de los hosts y las MVs, afectando al numero y tipología de las interacciones para cada par de MVs.

G. Técnica de asignación de host

Las técnicas de asignación deciden donde se instalarán las máquinas virtuales, ya sean nuevas o migradas. Por defecto, el host elegido siempre será el más potente que pueda contener la máquina vir-

[1] https://git.io/fjC0u

Algorithm 1 Algoritmo de selección de máquina virtual

Require: $\mathcal{H}$: Overloaded host
Ensure: $\mathcal{SMV}$: Selected MV
1: declare $\mathcal{MMV}$: migratable MVs, $\mathcal{CC}$: CPU correlation values, $\mathcal{NC}$: network correlation values, $\mathcal{HV}$: heuristic values
2: $\mathcal{MMV} \leftarrow getMigratableMVs(\mathcal{H})$
3: $\mathcal{CC} \leftarrow calculateCPUValues(\mathcal{H}, \mathcal{MMV})$
4: $\mathcal{NC} \leftarrow calculateNetworkValues(\mathcal{H}, \mathcal{MMV})$
5: $\mathcal{HV} \leftarrow calculateHVal(\mathcal{CC}, \mathcal{NC})$
6: $\mathcal{SMV} \leftarrow obtainBestMigratableMV(\mathcal{HV})$

TABLA II
Características de las máquinas virtuales

Tipo	Nº CPUs	MIPS	BW (Mbps)	Cantidad
Pequeña	1	1000	100	50
Grande	1	2000	100	50

TABLA III
Características de los hosts

Tipo	Modelo	Nº CPUs	MIPS	BW (Mbps)	Cantidad
Pequeño	HP ProLiant ML110 G4	2	1860	1000	6
Grande	HP ProLiant ML110 G5	2	2660	1000	4

TABLA IV
Modelo de energía de los hosts

Tipo	0%	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
Pequeño	86	89.4	92.6	96	99.5	102	106	108	112	114	117
Grande	93.7	97	101	105	110	116	121	125	129	133	135

TABLA V
Tipos de interacciones

Tipo	Proporción	Rango (%)
Baja	50%	[0, 20]
Media	30%	[20, 60]
Alta	20%	[60, 95]

TABLA VI
Parámetros de configuración de CloudSim

Parámetro	Valor
Traza CPU	PL 20110303
% interacciones internas	10%
% interacciones externas	0.05%
Número de switches	5
Número de hosts	10
Hosts asignación inicial	6
Número de máquinas virtuales	100
Intervalo de migraciones	12
BW máximo de las interacciones	10 Mbps
BW entre Host-MV	100 Mbps
BW entre Host-Switch	1000 Mbps
Límite saturación	70%
Tiempo de simulación	86400 s
Número de experimentos por técnica	30

tual sin saturarlo en el momento actual. También deciden cuándo un host está saturado.

La técnica de asignación utilizada es la IQR (InterQuartile Range) proporcionada por CloudSim. Para decidir cuándo un host está saturado, IQR utiliza los 12 últimos usos de CPU, indicando saturación cuando el valor obtenido por la eq. 4 sea mayor que 0.7.

$$IQR = Q3 - Q1 \tag{4}$$

H. Técnicas de selección

Esta técnica selecciona la MV candidata a ser migrada. Se han utilizado las técnicas por defecto de CloudSim con el objetivo de ser comparadas con la técnica de correlación propuesta en el presente trabajo. La técnicas utilizadas en la comparativa son las siguientes:

- Random Search (RS): de entre las máquinas virtuales candidatas a ser migradas, elige una aleatoriamente.
- Minimum Migration Time (MMT): Elige la máquina virtual que requiera menos memoria RAM.
- Minimum Utilization (MU): Elige la máquina virtual que ha solicitado la menor cantidad de MIPS a lo largo de toda la simulación.
- Maximum Correlation (MC): A partir de una

matriz con el porcentaje de utilización de cada MV de los 12 últimos instantes, se genera un modelo lineal con la transposición de esta matriz, generando finalmente una regresión lineal, eligiendo la MV con mayor correlación en relación al resto de MVs.

I. Período de migración

La Figura 5 muestra cómo el valor de las correlaciones evoluciona en función del número de muestras tomadas. Se ha elegido una ventana de 12 muestras ya que antes de 10 existe demasiada variabilidad, pero si se extiende demasiado, se tiende a obtener valores demasiado similares. En este experimento se permite realizar migraciones cada 12 monitorizaciones. Para simplificar la modelización, no existe ninguna penalización cuando se realiza una migración.

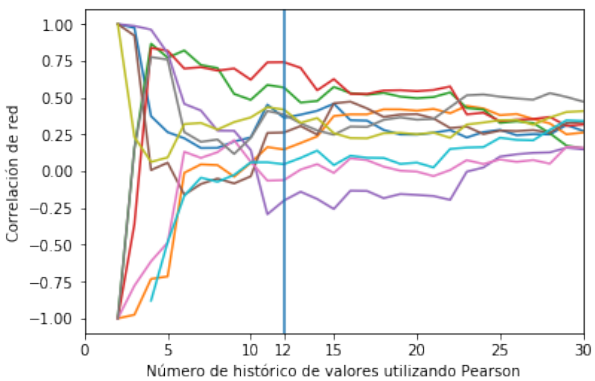


Fig. 5. Evolución de los valores de correlación a partir del número de muestras

VI. Resultados

A. Métricas

Las métricas analizadas en el presente trabajo son las siguientes:

- Ancho de banda total: El intercambio total de datos para cada interacción en cada instantánea.
- Datos saturados: La cantidad de datos que han superado la capacidad de un enlace.
- Hosts saturados: Hosts que han superado el 70% de utilización de CPU pero aún pueden satisfacer la demanda de CPU
- Hosts sobresaturados: Hosts que han alcanzado el 100% de utilización de CPU y no pueden satisfacer la demanda de CPU
- Hosts usados: la suma de los hosts en funcionamiento en cada instantánea
- MVs insatisfechas: MVs que no han recibido la cantidad de MIPS solicitada
- Porcentaje medio de MIPS no asignados: Ratio entre MIPS no asignados respecto a los asignados. Se ha realizado la media entre todos los hosts para cada instantánea.
- Migraciones totales: Suma del número de migraciones en cada instantánea
- Energía: Kilovatio hora (kWh) consumido por el datacenter

Las medias de estas métricas se muestran en la Tabla VII.

B. Consumo del Ancho de banda

Existe una relación entre el ancho de banda y su saturación en función de una óptima colocación de las MVs, cuanto menos ancho de banda, mayor concentración de éste en menos links, provocando mayor saturación. También puede existir saturación si MVs con mucha comunicación local son migradas a hosts lejanos, provocando más ancho de banda, que finalmente saturará los links centrales y más concurridos. Por lo tanto, se requiere un equilibrio entre estos dos factores. En la Figura 6 se puede observar que MU utiliza más ancho de banda, lo que provoca liberar links locales evitando la saturación de datos, mostrado en la Figura 7. Pearson consigue reducir el ancho de banda en un 2.5% y los datos saturados en un 5.7% sin contar con MU. Por lo tanto, observando las Figuras 6 y 7, Pearson es la técnica con un menor uso del ancho de banda, y además, consigue ser la segunda con menor saturación de datos.

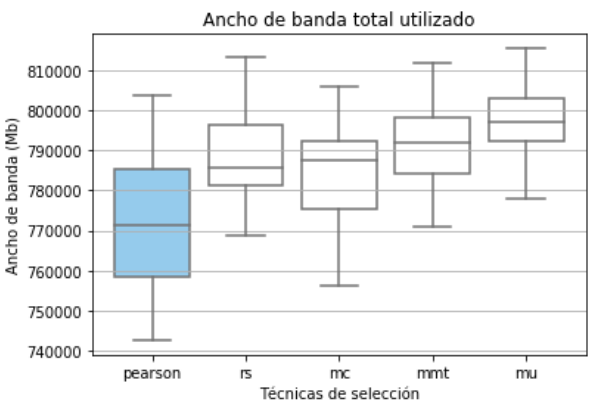


Fig. 6. Ancho de banda

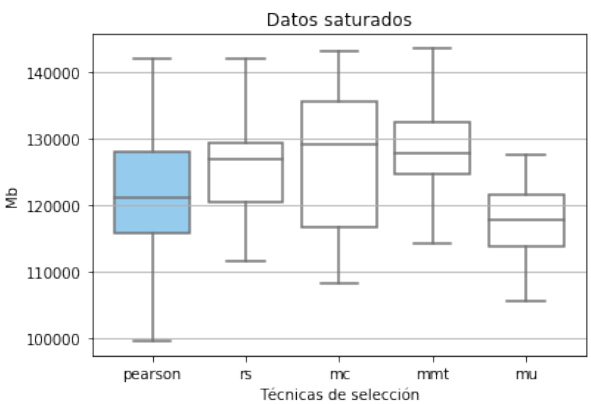


Fig. 7. Datos saturados

C. Consumo de CPU

Observando la Figura 8, Pearson consigue reducir de media un 6.9% el número de hosts utilizados res-

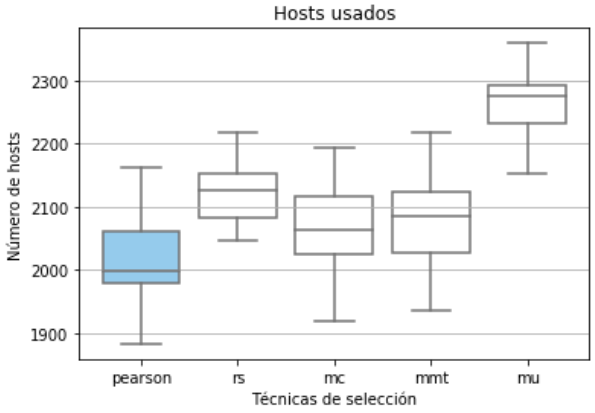


Fig. 8. Hosts usados

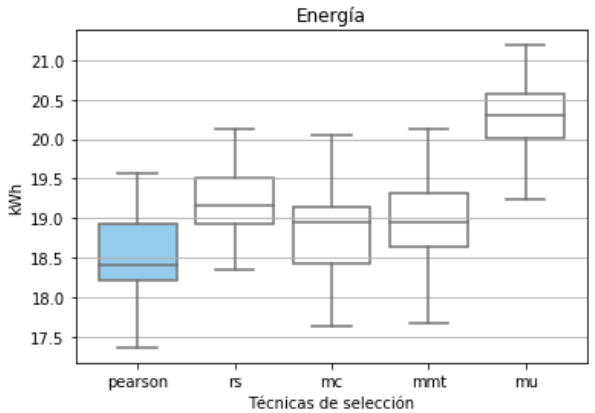


Fig. 9. Consumo energético/hora

pecto a las otras técnicas, siendo la única que requiere menos de 2000 hosts en total. Aunque esta disminución de hosts provoca un incremento de hosts saturados (Tabla VII-3) respecto a RS, MC y MMT (5.59% de media) y hosts sobresaturados (7.2% de media), pese a esta saturación, el número de MIPS asignados es muy estable (Tabla VII-9), con un incremento máximo de 0.14% (MC), y una reducción media de 2.23% respecto a MIPS no asignados (Tabla VII-10), siendo éstos una cantidad ínfima del total de MIPS de alrededor del 0.7%.

Aproximadamente el 40% de los hosts han estado saturados en algún momento (Tabla VII-2), las posibles explicaciones sobre éste valor serían un umbral de saturación alto junto a una variabilidad de CPU también elevada, y el número de instantáneas que tiene que esperar el sistema, pudiendo acontecer que en el momento de migración un host no esté saturado, pero en las anteriores instantáneas sí.

Se obtienen mejores resultados en relación a la energía (Figura 9) debido a un menor uso de hosts, aunque estos estén más saturados. Pearson consigue una mejora media del 4.77%.

D. Número de migraciones

La métrica más dispar es el número de migraciones, presentada en la Figura 10, obteniendo importantes mejoras respecto a las otras técnicas, reduciendo el número de migraciones en un 29.97% para RS, 14.1% para MC y 16.35% para MMT, y una gran mejora de 152.88% respecto MU.

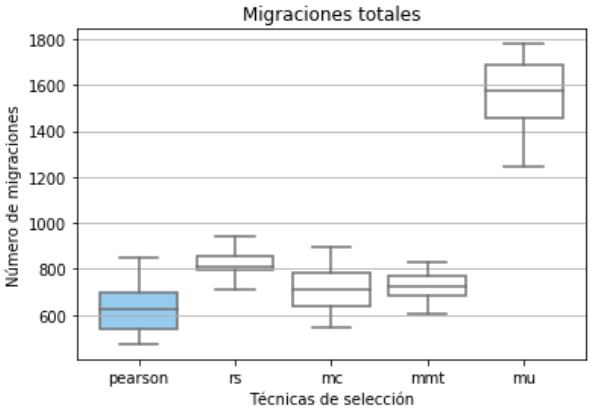


Fig. 10. Número de migraciones

E. Sumario

La heurística implementada utilizando el coeficiente de correlación de Pearson muestra una tendencia a limitar el número de migraciones, mejorando el uso de la red ya que las MVs con mayores comunicaciones internas se mantienen en el mismo host, siendo otras MVs menos comunicativas las que migren. Evitando la dispersión de MVs intercomunicadas se utiliza menos ancho de banda y se evita la saturación de datos en los switchs centrales. Aunque haya una mayor cantidad de hosts y MVs saturadas, el número de MIPS se mantiene estable, con menos de 1% de MIPS no asignados, por lo tanto, una mayor cantidad de hosts están trabajando al límite de sus capacidades. En cambio, el resto de técnicas tienen menos hosts saturados, pero éstos tienen una mayor cantidad de MIPS no asignados. Una menor cantidad de hosts utilizados permite obtener un consumo menor de energía, siendo Pearson la técnica que obtiene mejores resultados.

VII. Conclusiones

Realizando la media ponderada con el resto de técnicas, observamos que el uso del coeficiente de correlación de Pearson focalizado en la CPU y el ancho de banda obtiene una reducción media del ancho de banda en un 2.5% y una reducción media del consumo energético en un 5%, una reducción en la saturación de los datos de la red en un 5.7%, un incremento en la saturación de hosts de un 3.45% y una importante reducción del número de migraciones en un 53%. Por lo tanto, podemos afirmar que el uso de la heurística utilizando el coeficiente de correlación de Pearson permite reducir el número de migraciones ofreciendo resultados ligeramente mejores que el resto de técnicas.

A. Trabajo futuro

Se debe seguir trabajando en cómo las correlaciones pueden ser aprovechadas para realizar mejores migraciones. Por ejemplo, observando cual es el número óptimo de muestras, el número de máquinas

TABLA VII
MEDIAS DE LOS RESULTADOS

Índice	Métrica	pearson	rs	mc	mmt	mu
1	Ancho de banda total usado (Mb)	771272	785737	787709	791914	797151
2	Datos totales saturados (Mb)	121075	126948	129147	127862	117725
3	Hosts saturados totales	804	785	771	773	776
4	Energía (kWh)	18.415	19.162	18.943	18.953	20.293
5	Número de migraciones	624	811	712	726	1578
6	MVs totales sobresaturadas	1457	1388	1361	1403	1531
7	Hosts totales sobresaturados	72	69	66	71	74
8	Hosts totales usados	1999	2127	2063	2084	2276
9	MIPS totales asignados	5130502	5125513	5138098	5134598	5129031
10	MIPS totales no asignados	37888	37075	34917	38145	38026
12	Porcentaje medio de MIPS no asignados	0.745	0.734	0.700	0.737	0.734

virtuales y añadir peso a las correlaciones en función de la utilización.

Además, nos planteamos abordar las siguientes líneas de trabajo futuro:

- Implementar otras técnicas de selección de la literatura con el fin de completar el análisis con las últimas propuestas.
- Modelizar el tiempo de migración con el fin de incluir el coste de las migraciones en la toma de decisiones.
- Desarrollar nuevas meta-heurísticas para decidir si una MV debe o no ser migrada a partir de las correlaciones de CPU y ancho de banda.

AGRADECIMIENTOS

Este trabajo ha sido posible gracias a MEIyC-Spain TIN2017-84553-C2-2-R.

REFERENCIAS

[1] Sareh Fotuhi Piraghaj, Amir Vahid Dastjerdi, Rodrigo N Calheiros, and Rajkumar Buyya, "Containercloudsim: An environment for modeling and simulation of containers in cloud data centers," *Software: Practice and Experience*, vol. 47, no. 4, pp. 505–521, 2017.

[2] Manoel C Silva Filho, Raysa L Oliveira, Claudio C Monteiro, Pedro RM Inácio, and Mário M Freire, "Cloudsim plus: a cloud computing simulation framework pursuing software engineering principles for improved modularity, extensibility and correctness," in *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*. IEEE, 2017, pp. 400–406.

[3] Lucio A Rocha and Fábio L Verdi, "Milpflow: A toolset for integration of computational modelling and deployment of data paths for sdn," in *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*. IEEE, 2015, pp. 750–753.

[4] Bhathiya Wickremasinghe and Rajkumar Buyya, "Cloudanalyst: A cloudsim-based tool for modelling and analysis of large scale cloud computing environments," *MEDC project report*, vol. 22, no. 6, pp. 433–659, 2009.

[5] Xiang Li, Xiaohong Jiang, Kejiang Ye, and Peng Huang, "Dartcsim+: Enhanced cloudsim with the power and network models integrated," in *2013 IEEE Sixth International Conference on Cloud Computing*. IEEE, 2013, pp. 644–651.

[6] Alberto Núñez, Jose L Vázquez-Poletti, Agustin C Caminero, Gabriel G Castañé, Jesus Carretero, and Ignacio M Llorente, "icancloud: A flexible and scalable cloud infrastructure simulator," *Journal of Grid Computing*, vol. 10, no. 1, pp. 185–209, 2012.

[7] Zahra Nikdel, Bing Gao, and Stephen W Neville, "Dockersim: Full-stack simulation of container-based software-as-a-service (saas) cloud deployments and environments," in *2017 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM)*. IEEE, 2017, pp. 1–6.

[8] Theo Lynn., Huanhuan Xiong., Dapeng Dong., Bilal Momani., George Gravvanis., Christos Filelis-Papadopoulos., Anne Elster., Malik Muhammad Zaki Murtaza Khan., Dimitrios Tzovaras., Konstantinos Giannoutakis., Dana Petcu., Marian Neagul., Ioan Dragan., Perumal Kuppudayar., Suryanarayanan Natarajan., Michael McGrath., Georgi Gaydadjiev., Tobias Becker., Anna Gourinovitch., David Kenny., and John Morrison., "Cloudlightning: A framework for a self-organising and self-managing heterogeneous cloud," in *Proceedings of the 6th International Conference on Cloud Computing and Services Science - Volume 1: CLOSER,*. INSTICC, 2016, pp. 333–338, SciTePress.

[9] Douglas G Bonett and Thomas A Wright, "Sample size requirements for estimating pearson, kendall and spearman correlations," *Psychometrika*, vol. 65, no. 1, pp. 23–28, 2000.

[10] Nian Shong Chok, *Pearson's versus Spearman's and Kendall's correlation coefficients for continuous data*, Ph.D. thesis, University of Pittsburgh, 2010.

[11] Jan Hauke and Tomasz Kossowski, "Comparison of values of pearson's and spearman's correlation coefficients on the same sets of data," *Quaestiones geographicae*, vol. 30, no. 2, pp. 87–93, 2011.

[12] Joost CF de Winter, Samuel D Gosling, and Jeff Potter, "Comparing the pearson and spearman correlation coefficients across distributions and sample sizes: A tutorial using simulations and empirical data.," *Psychological methods*, vol. 21, no. 3, pp. 273, 2016.

[13] Sonu Choudhary and Abhay Kothari, "Green data center using spearmans ranking algorithm," *International Journal of Computer Science and Information Technologies*, vol. 6, no. 2, 2015.

[14] Seyedhamid Mashhadi Moghaddam, Sareh Fotuhi Piraghaj, Michael O'Sullivan, Cameron Walker, and Charles Unsworth, "Energy-efficient and sla-aware virtual machine selection algorithm for dynamic resource allocation in cloud data centers," in *2018 IEEE/ACM 11th International Conference on Utility and Cloud Computing (UCC)*. IEEE, 2018, pp. 103–113.

[15] Rachael Shaw, Enda Howley, and Enda Barrett, "A predictive anti-correlated virtual machine placement algorithm for green cloud computing," in *2018 IEEE/ACM 11th International Conference on Utility and Cloud Computing (UCC)*. IEEE, 2018, pp. 267–276.

[16] Nilson Moraes Filho, "Improving Load Balancing in Virtualized Environments using Pearson's Correlation," M.S. thesis, UFSCar, Sorocaba, São Paulo, Brasil, 2018.

[17] Anton Beloglazov and Rajkumar Buyya, "Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers," *Concurrency and Computation: Practice and Experience*, vol. 24, no. 13, pp. 1397–1420, 2012.

[18] KyoungSoo Park and Vivek S Pai, "Comon: a mostly-scalable monitoring system for planetlab," *ACM SIGOPS Operating Systems Review*, vol. 40, no. 1, pp. 65–74, 2006.

Tecnologías clúster, plataformas distribuidas, BigData y Deep Learning

Uso de Composiciones Paralelas de Alto Nivel para el reconocimiento de secuencias ADN mediante una Red Neuronal Convolucional

Mario Rossainz-López[1], Sarahi Zúñiga-Herrera[1], Ivo Pineda-Torres[1], Manuel I. Capel-Tuñón[2]

Resumen— **El presente trabajo propone con el uso de la programación paralela estructurada la implementación del patrón de comunicación entre procesos denominado Pipeline como una Composición Paralela de Alto Nivel (CPAN) que represente una red neuronal convolucional utilizada para resolver un problema específico de secuencias de ADN. Se muestra el CPAN denominado Pipeline-CNN, que representa una red neuronal convolucional a través de los tres tipos de objetos paralelos que lo componen: un objeto manager, uno o más objetos stage y un objeto collector. El objeto manager representa el CPAN en sí mismo y hace de él una abstracción encapsulada que oculta su estructura interna, los objetos stage son objetos de propósito particular encargados de encapsular una interfaz de tipo cliente-servidor, que se establece entre dichos stages y los llamados objetos esclavos (estos objetos no participan activamente en la composición del CPAN pues son entidades externas que contienen el algoritmo secuencial que constituye la solución de un problema dado), y el objeto colector que es un objeto encargado de almacenar los resultados recibidos. Un CPAN proporciona la interconexión necesaria para implementar la semántica del patrón de comunicación definido, que en este caso es un pipeline. Para mostrar la utilidad y el rendimiento del CPAN Pipeline-CNN implementado, se utilizó en un caso de estudio particular: el reconocimiento de secuencias de ADN de una base de datos con 4 tipos de virus de la hepatitis C (tipo 1, 2, 3 y 6) tomado del repositorio disponible en la página de ViPR. Con la red neuronal convolucional implementada como CPAN, los límites entre exones e intrones se reconocieron mediante una representación gráfica de las secuencias de ADN del virus y se utilizaron algoritmos de aprendizaje profundo para el aprendizaje del CPAN Pipeline-CNN. Se muestra además la metodología utilizada para la representación de secuencias de cadenas de ADN en imágenes y el entrenamiento del CPAN Pipeline-CNN con las imágenes para obtener una clasificación de los tipos de virus de la hepatitis C. Los resultados de esta clasificación se obtuvieron en términos de porcentajes de precisión de entrenamiento y precisión de validación, así como resultados de rendimiento en términos de aceleración de 1000 a 4000 pasos de entrenamiento con 2, 4, 8, 16 y 32 procesadores exclusivos en una máquina paralela de hasta 64 procesadores con memoria compartida distribuida.**

Palabras Clave—**Composiciones Paralelas de Alto Nivel, CPAN, Red Neuronal Convolucional, CNN, Transferencia de Aprendizaje Profundo, Secuencias ADN, Objetos Paralelos, Programación Paralela Estructurada.**

[1]Universidad Autónoma de Puebla, Avenida. San Claudio y 14 Sur, San Manuel, Puebla, Puebla, 72000, México. E-mail: {rossainz, ipineda}@cs.buap.mx, zarahi.suhe@gmail.com

[2]DPT. Ingeniería de Software, Colegio de Informática y Telecomunicaciones, Universidad de Granada, ETSIIT Calle Periodista Daniel Saucedo Aranda s/n, 18071 Granada, España. E-mail: manuelcapel@ugr.es

I. Introducción

Las redes neuronales convolucionales o CNN son redes neuronales multicanal, su principal ventaja es que cada parte de la red está entrenada para realizar una tarea, esto reduce significativamente el número de capas ocultas, por lo que el entrenamiento es más rápido [1]. Las redes neuronales convolucionales son muy poderosas para todo lo que tiene que ver con el análisis de imágenes. Sin embargo, su uso no está restringido ello, pues también se puede aplicar al reconocimiento de voz o a la clasificación de oraciones con las transformaciones necesarias con respecto al tipo de datos de entrada del problema a resolver [1, 2]. Particularmente en el análisis de imágenes, una red neuronal convolucional es una red multicapa que consiste en capas convolucionales y de reducción alternas, justamente formando una arquitectura pipeline. En la convolución, las operaciones de productos y sumas se llevan a cabo entre la capa de inicio (stage inicial) y los n filtros (n-stages) lo que genera un mapa característico o matriz. Las características extraídas corresponden a cada ubicación posible del filtro en la imagen original. La ventaja es que el mismo filtro (neurona) sirve para extraer la misma característica en cualquier parte de la entrada, con esto es posible reducir el número de conexiones y el número de parámetros a entrenar en comparación con una red multicapa de conexión total [1, 2]. En la reducción, el número de parámetros se reduce al mantener las características más comunes. La última capa de esta red es una capa de ordenamiento que tendrá tantas neuronas como el número de clases a predecir. Sin embargo, una de las desventajas de las redes neuronales es la gran cantidad de tiempo que se necesita para el entrenamiento. Una forma directa de reducir este tiempo es paralelizar los algoritmos de aprendizaje. No obstante, los algoritmos no siempre pueden ser paralelizados de una manera simple, y,

además, la cantidad de comunicación entre procesadores o procesos hace que la mayoría de las versiones paralelas de estos algoritmos solo pueden ejecutarse correctamente en computadoras paralelas. Es por eso que en este trabajo se propone una paralelización de una red neuronal convolucional bajo el modelo de composiciones paralelas de alto nivel o CPANs como una propuesta original y útil para obtener un buen rendimiento en el uso de una CNN dentro de un problema concreto. Los CPANs son patrones paralelos de comunicación bien definidos y lógicamente estructurados que, una vez identificados en términos de sus componentes y de su esquema de comunicación, se llevan a la práctica como constructos añadidos a un lenguaje de programación orientado a objetos como abstracciones de alto nivel en las aplicaciones del usuario, dentro de un entorno o ambiente de programación particular [3]. Las estructuras de interconexión de procesos tales como los árboles, las granjas o los cauces se pueden construir en sus versiones paralelas utilizando CPANs, dentro del entorno de trabajo de los Objetos Paralelos que se utilizan para detallar la estructura de una implementación de un CPAN tal como lo propone [4]. Un enfoque estructurado para la programación paralela se basa en el uso de patrones de comunicación/interacción que son estructuras predefinidas de los procesos de la aplicación del usuario [5]. En ese sentido, el enfoque de paralelismo estructurado proporciona la abstracción del patrón de interacción y la descripción de las aplicaciones a través del CPAN y que son capaces de implementar el patrón ya mencionado. La encapsulación de un CPAN debe seguir el principio de modularidad y debe proporcionar una base para obtener una reutilización efectiva del comportamiento paralelo a implementar. Cuando existe la posibilidad de lograr esto, se construye un patrón paralelo genérico, que a su vez proporciona una posible implementación de la estructura de interacción entre los procesos de la aplicación, independientemente de la funcionalidad de estos procesos. Se pueden identificar varios patrones paralelos de interconexión significativos y reutilizables en múltiples aplicaciones y algoritmos [6] que han dado como resultado una amplia biblioteca de patrones de comunicación entre procesos concurrentes, como son los CPANS y cuyos detalles se encuentran en [7, 8]. En el presente trabajo, se propone la implementación del CPAN Pipeline-CNN, que representa una red neuronal convolucional que utiliza transferencia de aprendizaje profunda como una estrategia de aprendizaje para la red neuronal y su uso en el reconocimiento de secuencias de ADN de una base de datos con 4 tipos del virus de la hepatitis C (tipos 1, 2, 3 y 6) tomado del repositorio disponible en la página de ViPR. El conjunto de secuencias de ADN utilizado es el conjunto de datos de la base de datos Molecular (Secuencias de genes de unión de empalme) que tiene 3190 secuencias, disponibles en la página de la UCI, con tres clases de secuencias: límite de exón-intrón, límite de intro-exón y cualquier otro. Para el uso de las secuencias de ADN se diseñó un método de representación donde cada base nitrogenada se representa en escala de grises para formar una imagen. Las imágenes generadas se utilizaron para entrenar la red neuronal convolucional CPAN Pipeline-CNN. Se muestran los resultados tanto de la clasificación llevada a cabo por el CPAN Pipeline-CNN en términos de precisión de entrenamiento y precisión de validación, como el rendimiento paralelo en su ejecución, obteniendo medidas de la ley de Amdahl y de aceleración en una computadora paralela con 32 CPU-SET exclusivos.

II. Definición de una Composición Paralela de Alto Nivel

Usando el paradigma de la Orientación a Objetos el objetivo es representar cualquier tipo de patrones paralelos de comunicación entre los procesos de una aplicación o algoritmo paralelo/distribuido como clases. Un CPAN proviene de la composición de un conjunto de objetos de tres tipos: el objeto Manager, los objetos stage y el objeto Collector. El objeto manager representa al CPAN en sí mismo y hace de él una abstracción encapsulada que oculta su estructura interna (Fig.1). El manager controla las referencias del objeto Collector y varios objetos Stage, que representan los componentes del CPAN y cuya ejecución se lleva a cabo en paralelo y debe ser coordinada por el propio manager [9]. Los objetos Stage son objetos de propósito específico, encargados de encapsular una interfaz tipo cliente-servidor que se establece entre el manager y los objetos esclavos (objetos que no son activamente participativos en la composición del CPAN, sino que se consideran entidades externas que contienen el algoritmo secuencial que constituye la solución de un problema dado) y el objeto Collector encargado de almacenar en paralelo los resultados que le lleguen de los objetos stage que tenga conectados. Durante el servicio de una petición, el flujo de control dentro de los stages de un CPAN depende del patrón de comunicación implementado.

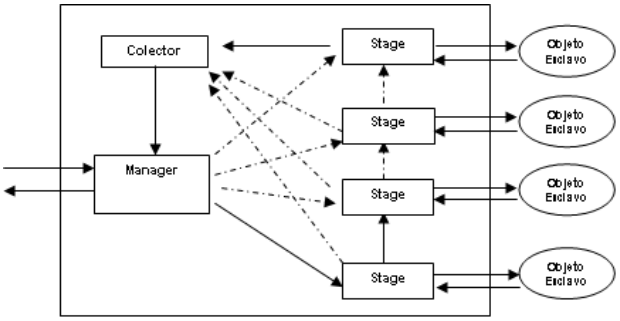


Fig. 1. Estructura de una Composición Paralela de Alto Nivel

Los objetos manager, collector y stages se engloban dentro de la definición de Objeto Paralelo (PO) [10]. Los Objetos Paralelos son objetos activos que tienen capacidad de ejecución en sí mismos. Las aplicaciones dentro del modelo PO pueden explotar tanto el paralelismo entre objetos (inter-object) como el paralelismo interno de ellos (intra-object) [11, 12]. Un objeto PO tiene una estructura similar a la de un objeto en C++ o JAVA, pero además incluye una política de planificación que especifica la

forma de sincronizar una o más operaciones de la clase del objeto susceptibles de invocarse en paralelo [12]. Los modos de comunicación utilizados en los CPANs son: el modo de comunicación síncrono, asíncrono y futuro asíncrono. La descripción de estar formas de comunicar a los procesos se encuentran en [13, 14]. Cuando se producen peticiones paralelas de servicio en un CPAN, es necesario disponer de mecanismos de sincronización para que los objetos gestionen varios flujos de ejecución concurrentemente y se garantice la consistencia de los datos que se están procesando. En un CPAN se pueden utilizar las restricciones MAXPAR o paralelismo máximo, MUTEX o exclusión mutua y SYNC sincronización del tipo productor-consumidor, para la correcta programación de sus métodos [15].

La Fig. 1 muestra el modelo CPAN en su forma abstracta. La caja que engloba a los componentes representa el CPAN encapsulado. Las cajas internas representan objetos compuestos (*collector*, *manager* y objetos *stages*), en tanto que los óvalos son los objetos esclavos asociados a los *stages*. Las líneas continuas dentro del *CPAN* suponen que al menos debe existir una conexión; lo mismo sucede entre los *stages* y el objeto *collector*. Las líneas punteadas significan que puede haber más de una conexión entre los componentes.

A. Construcción de Patrones de Comunicación entre Procesos como CPANs

Actualmente se cuenta con una biblioteca de clases que proporciona al programador los patrones de comunicación entre procesos más comúnmente utilizados en su representación de CPANs: El farm, el pipeline y el árbol de procesos (particularizado a árboles binarios). Las fig. 2, 3 y 4 muestran los respectivos modelos como Composiciones Paralelas de Alto Nivel. Estos modelos son abstractos, es decir, el programador debe adaptarlos al problema que se esta tratando de resolver haciendo uso de las propiedades del paradigma de la orientación a objetos tales como la herencia o el polimorfismo. La estructura de la biblioteca de clases se muestra en el diagrama de clases de la fig. 5. Con el conjunto básico de clases del modelo de programación de *PO* se pueden construir *CPANS* concretos.

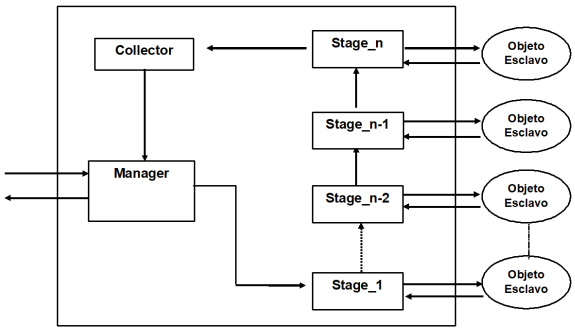


Fig 2. El Pipeline como Composición Paralela de Alto Nivel

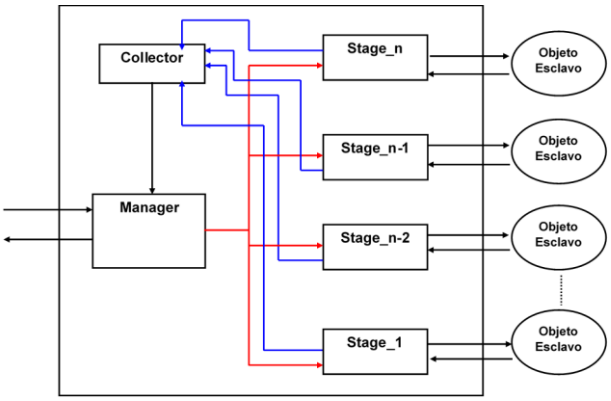


Fig 3. El Farm como Composición Paralela de Alto Nivel

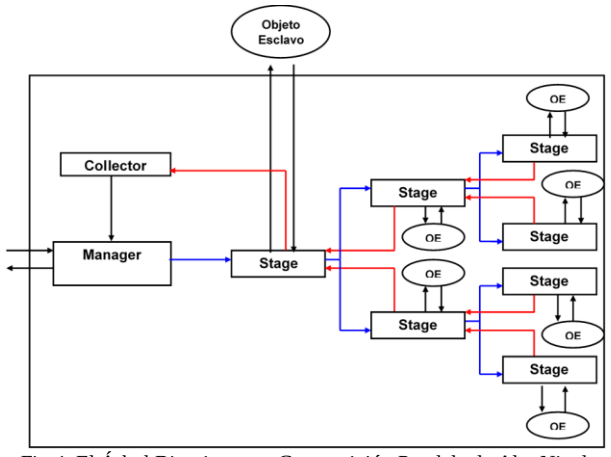


Fig 4. El Árbol Binario como Composición Paralela de Alto Nivel

Fig 5. Diagrama de clases de la biblioteca de los CPANs

Para construir un *CPAN*, primero se debe tener claro el comportamiento paralelo que se necesita implementar en la aplicación que se pretenda desarrollar, de tal forma que el *CPAN* en sí mismo pueda representar lo más ajustadamente dicho patrón de comportamiento. En relación con los patrones de comunicación que se establecen entre los procesos de una aplicación paralela y

distribuida existen varias posibilidades de interacción: los *farms* (o granjas de procesos), los *pipes* (o cauces), los trees (o árboles), los *cubes* (o cubos), los *grids* (o mallas), las *matrices de procesos*, etc. De tal forma que saber qué patrón se adapta mejor a una aplicación determinada constituye una decisión importante de diseño que no puede ser totalmente automatizada. Una vez identificado el comportamiento paralelo de la aplicación en desarrollo, el segundo paso consiste en elaborar un bosquejo gráfico de su representación, con los detalles concretos de ésta: número de nodos, canales de conexión entre estos, etc., como un documento previo al diseño detallado en el que se detallará posteriormente el procesamiento paralelo del sistema objetivo. También sirve para representar sus características generales y permitirá después definir su representación como CPANS siguiendo el modelo propuesto en la fig 1. Cuando ya se tiene concretizado el modelo de un *CPAN*, que define un patrón paralelo específico, digamos, por ejemplo, un *tree*, o alguno de los anteriormente mencionados, el paso siguiente será realizar su definición sintáctica y semántica [16]. Traduciéndose finalmente la definición sintáctica a un *CPAN*, que programado en el entorno de programación más adecuado para su implementación paralela ha de verificarse para que la semántica resultante sea la correcta; es conveniente probarlo con varios ejemplos distintos para demostrar su genericidad, así como observar el rendimiento de las aplicaciones que lo incluyan como un componente software.

Los modelos de CPANs de las fig. 2, 3 y 4 han sido utilizados y adaptados a patrones de farms, pipeline y trees particulares, de problemas que han sido resueltos con ellos, por ejemplo: problemas de ordenación, búsqueda y optimización, problemas NP-Completos tales como el del Agente Viajero, problemas de simulación como el movimiento y atracción de partículas en el espacio y más recientemente problemas que tienen que ver con encontrar las secuencias ADN en la construcción de GNOMAS.

III. Redes Neuronales Convolucionales (CNN)

En los últimos años, el campo del aprendizaje automático ha progresado enormemente al abordar los problemas de clasificación, identificación y reconocimiento de patrones. En particular, se ha encontrado que un tipo de modelo llamado red neuronal convolucional o CNN logra un desempeño razonable en tareas de reconocimiento visual de hardware, igualando o superando el desempeño humano en algunos dominios [17]. Una CNN es un algoritmo para el aprendizaje automático en el que un modelo aprende a realizar tareas de clasificación directamente desde imágenes, videos o sonidos. Las CNN son especialmente útiles para localizar patrones en imágenes con el fin de reconocer objetos, caras y escenas. Aprenden directamente de los datos de la imagen, utilizando patrones para clasificarlas y eliminar la necesidad de una extracción manual de características. Para que una CNN aprenda, se utilizan modelos de aprendizaje profundo. El más común es Inseption-V3, diseñado para el Reconocimiento Visual. Ésta es una tarea estándar en visión artificial, donde los modelos intentan clasificar imágenes completas en una red de más de 1000 clases de imágenes. Este modelo está disponible en TensorFlow, que es una herramienta para el aprendizaje automático. TensorFlow está diseñado principalmente para modelos de redes neuronales profundas [18]. Los modelos modernos de reconocimiento de imágenes tienen millones de parámetros; entrenarlos desde cero requiere una gran cantidad de datos etiquetados para entrenar la CNN y mucha potencia de cálculo. Por lo tanto, una de las desventajas de las redes neuronales es la gran cantidad de tiempo necesario para el aprendizaje. Una forma directa de reducir este tiempo es paralelizar los algoritmos de aprendizaje. Sin embargo, los algoritmos no siempre se pueden paralelizar de una manera sencilla y, además, la cantidad de comunicación entre los procesos hace que la mayoría de las versiones paralelas de estos algoritmos solo se puedan ejecutar en computadoras paralelas [19]. La transferencia de aprendizaje es una técnica rápida que toma una pieza de un modelo que ya ha sido entrenado en una tarea relacionada y lo reutiliza en un nuevo modelo. La Fig. 6 (tomada de https://www.slideshare.net/ManuelRodrigoCabello/deep-learning-python-c-y-azure) muestra un ejemplo de una CNN; los filtros se aplican a cada entrenamiento de la imagen con diferentes resoluciones, y la salida de cada imagen convolucionada se usa como entrada para la siguiente capa que genera un patrón de comunicación pipeline y que puede ser paralelizado [18, 19].

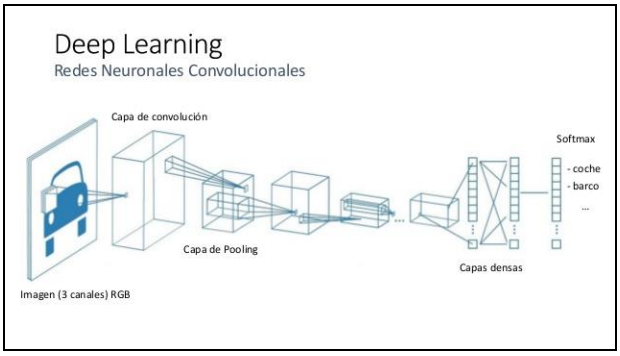


Fig 6. Red Neuronal Convolucional

La técnica transferencia de aprendizaje es efectiva para muchas aplicaciones, funciona con cantidades moderadas de datos de entrenamiento (miles, no millones de imágenes etiquetadas) y se puede ejecutar secuencialmente en minutos u horas. En este trabajo, se muestra la paralelización de una red neuronal convolucionada bajo el modelo CPAN. El CPAN Pipeline es adaptado a un modelo de red neuronal convolucional con la técnica de transferencia de aprendizaje lo que permite su ejecución en ordenadores paralelos o computadores con GPUs.

IV. Representación de una CNN como un CPAN usando Transferencia de Aprendizaje Profundo

Una red convolucional es una red neuronal, con las funciones de activación o las capas totalmente conectadas, pero además con dos conceptos nuevos: la capa convolucional y la capa de agrupación o muestreo. Las arquitecturas de las redes convolucionales se construyen apilando estos elementos, es por eso que de acuerdo con el uso de memoria y cómputo de una red neuronal para el procesamiento de imágenes [20], es útil y apropiado representarla a través de un CPAN Pipeline. Para el entrenamiento de una red neuronal convolucional, se utilizó la transferencia de aprendizaje mediante la extracción de descriptores profundos como una forma de entrenamiento y validación de la red neuronal en el conjunto de imágenes del problema específico a resolver. De esta manera obtenemos el CPAN Pipeline-CNN que se muestra en la fig. 7, y que ayudará a resolver el caso de estudio que se muestra en las siguientes secciones de este artículo.

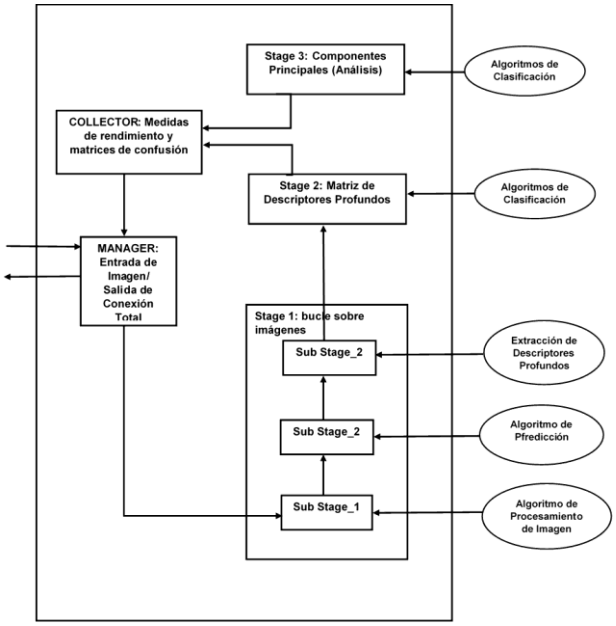


Fig 7. El CPAN Pipeline-CNN

En el CPAN Pipeline-CNN de la fig. 7, en la primera capa convolucional, las neuronas que forman la CNN se conectan a una parte de la imagen de entrada proporcionada por el usuario a través del objeto Manager del CPAN y no a la totalidad. Cuando se concatenan varias capas convolucionales a una parte de la salida de una capa, ciertas neuronas de la siguiente capa están conectadas, pero no todas ellas forman la segunda capa. Esto se lleva a cabo en la primera etapa del CPAN Pipeline-CNN (bucle sobre imágenes). Después de varias capas convolucionales concatenadas, se obtienen detalles sobre las características de la imagen, por ejemplo, formas o colores. En cada capa convolucional, los mapas de características se apilan. Un mapa de características es una capa donde todas las neuronas usan el mismo filtro y comparten parámetros característicos. En cada capa convolucional, se aplican tantos filtros como mapas de características se apilan en ella. La transferencia de aprendizaje en el CPAN Pipeline-CNN se produce al describir descriptores profundos en la sub-etapa2 del modelo. Este ajuste se produce mediante el entrenamiento y la validación del conjunto de imágenes del problema específico que se resuelve. El trabajo se completa en las etapas 2 y 3 del CPAN Pipeline-CNN ejecutando los algoritmos de clasificación asociados a los objetos esclavos, para generar la matriz de descriptores profundos y el análisis de los componentes principales, obteniendo resultados que son enviados al objeto Collector tales como las medidas de rendimiento y las matrices de confusión (ver fig. 6 y fig. 7). La ejecución interna de los objetos paralelos del CPAN incluidos el Manager, el Collector y los stages, así como el paralelismo entre ellos, hacen que la solución del problema específico que se resuelva obtenga un mejor rendimiento que su contraparte secuencial, sobre todo cuando se trabaja con una gran cantidad de imágenes del orden de cientos de miles bajo una arquitectura de hardware también paralela.

V. Reconocimiento de Secuencias ADN usando el CPAN Pipeline-CNN

Los procesos de predicción de genes son aquellos que, dentro del área de la biología computacional, se utilizan para la identificación algorítmica de fragmentos de secuencias de ADN [21], y que son biológicamente funcionales. La identificación de genes es un área importante para entender el genoma de una especie una vez que éste ha sido secuencializado. El ADN está compuesto por cuatro moléculas llamadas nucleótidos o bases nitrogenadas: adenina, timina, guanina y citosina [22]. Una secuencia de ADN se compone de un alfabeto que contiene las letras de las cuatro bases nitrogenadas (fig. 8).

GTAGTCATGTTGAAAAACTTACGAGTAAATTACGTTGTCGA
GGGCGTGCAAGTAGCGCAACCCGTGACAAGCGCAAATTCG
GAAGTAT**ACGCCAA**TCTACCGCCTCCCGTACCCGCGGAGAC
GTATCAAACCGACGAAGATTACGAGGAAGATGACGGAGG
GTGGGC

Fig 8. Una secuencia de ADN

Una secuencia de ADN puede definir las características de un organismo vivo que contiene toda la información genética en unidades de herencia llamadas genes. Las uniones de empalme son puntos en una secuencia de ADN en la que el ADN "inútil" se elimina durante el proceso de creación de proteínas en organismos superiores. El problema entonces es reconocer con una secuencia de ADN los límites entre los exones (las partes de la secuencia de ADN que se retienen después del empalme) y los intrones (las partes de la secuencia de ADN que se cortan). Este problema consiste en dos subtareas: reconocimiento de límites de exón/intrón (llamado "EI" o

donante) y reconocimiento de límites de intrón/exón (sitios "IE" o aceptador), [23]. Ambas tareas son complicadas ya que no hay una secuencia estándar para reconocer intrones y exones por lo que es interesante diseñar herramientas que nos ayuden a identificarlos y clasificarlos. Para mejorar la representación de una cadena de ADN se utilizan secuencias que pueden transformarse en una representación de valores numéricos o alfabéticos: A (adenina), T (timina), G (guanina) y C (citosina), [24], como muestra la fig. 8. Sin embargo, la representación de grandes cantidades de información como secuencias de ADN no facilita su análisis matemático lo que crea la necesidad de encontrar nuevas formas de representar la información. Se presenta como caso de estudio la generación de imágenes que representan secuencias de ADN para ser analizadas mediante técnicas de aprendizaje profundo utilizando como herramienta para ello la creación de una red neuronal convolucional bajo la propuesta del CPAN Pipeline-CNN que se mostró previamente en la presente investigación y poder así clasificar las imágenes. La idea es convertir las secuencias de ADN en representaciones gráficas para entrenar el CPAN Pipeline-CNN. Hay que recordar que las CNN se utilizan para el reconocimiento de patrones y clasificación de imágenes. Las secuencias de ADN están representadas por las letras: A-adenina, G-guanina, C-citosina y T-timina, sin embargo, la CNN propuesta no se creó para procesar información con este formato, por el contrario, se diseñó una representación gráfica de las secuencias para que el CPAN Pipeline-CNN pudiera ser de utilidad.

A. Caso de Estudio: Secuencias de ADN del Virus de la Hepatitis-C

Se utilizaron 1847 secuencias de ADN de una base de datos con 4 tipos de virus de la hepatitis C (tipo 1, 2, 3 y 6) tomados del repositorio disponible en la página de ViPR y un conjunto de secuencias de ADN de la base de datos Molecular (secuencias genéticas de unión de empalmes) Conjunto de datos que tiene 3190 secuencias, disponibles en la página de la UCI, con tres clases de secuencias: límite de exón-intrón, límite de intrón-exón y ninguno. La metodología utilizada fue la siguiente:

1. Se asignó un color de escala de grises a cada una de las letras de la secuencia de ADN (ver la tabla I), que va de un valor de 0 = negro a un valor de 1 = blanco, de modo que los colores intermedios son tonos de gris para mostrar un mejor contraste.
2. Se creó la imagen que representa las secuencias de ADN: se utilizó una matriz de dimensión 60 X 60, donde el valor 60 coincide con el número de bases nitrogenadas de todas las secuencias de la base de datos.

TABLA I. ESCALA DE GRISES DE BASES NITROGENADAS

Base Nitrogenada	*Escala de gris*
A	0
C	0.3
G	0.7
T	1

Cada secuencia se colocó en la primera fila y se copió en el resto de las filas hasta que llegar a un total de 60 (ver fig. 9). El resultado es una imagen de barras en escala de grises como la que se muestra en la Fig. 10. Cada una de las imágenes obtenidas es específica de cada instancia de la base de datos que se muestra en la Fig 9. En total, se obtuvieron 3190 imágenes.

```
1  CCAGCTGCATCACAGGAGGCCAGCGAGCAGGTCTGTTCCAAGGGCCTTCGAGCCAGTCTG
2  CCAGCTGCATCACAGGAGGCCAGCGAGCAGGTCTGTTCCAAGGGCCTTCGAGCCAGTCTG
3  CCAGCTGCATCACAGGAGGCCAGCGAGCAGGTCTGTTCCAAGGGCCTTCGAGCCAGTCTG
.
.
.
60 CCAGCTGCATCACAGGAGGCCAGCGAGCAGGTCTGTTCCAAGGGCCTTCGAGCCAGTCTG
```

Fig 9. Secuencia de ADN a ser codificada

Fig 10. Imagen de barras de una instancia de secuencia de ADN

3. Con las imágenes representativas de cada secuencia, se entrenó al CPAN Pipeline-CNN. Este modelo de CPAN se basa en el modelo CNN InceptionV3 con transferencia de aprendizaje profundo para categorizar el reconocimiento de tres clases de secuencias de ADN: reconocimiento de límites de exón / intrón (sitios EI), reconocimiento de límites intrón / exón (sitios IE) y reconocimiento de Ninguno de los dos anteriores (N).
4. Con la herramienta de software TensorFlow se construyó un modelo de clasificación y se colocó como un objeto esclavo dentro del CPAN Pipeline-CNN para ser paralelizado (ver fig. 7). Esto se logró categorizando el reconocimiento de una base de datos con cuatro clases de secuencias de ADN: virus de la hepatitis C tipo 1, 2, 3 y 6 y el reconocimiento de otra base de datos con tres tipos de límites: EI, IE y N.
5. Con el CPAN Pipeline-CNN se entrenaron las últimas capas de la red con instancias obtenidas de las bases de datos en 4000 pasos. Primero, el CPAN Pipeline-CNN fue entrenado para clasificar los 4 tipos de virus de la hepatitis y luego el entrenamiento se realizó con 2 clases: EI e IE y finalmente con todas las clases de la base de datos: EI, IE y N para comparar los resultados de las dos últimas neuronas.
6. Finalmente, se obtuvieron los resultados de clasificación y análisis de rendimiento del modelo del CPAN propuesto.

VI. Resultados y Rendimiento

El equipo de cómputo utilizado para el entrenamiento del CPAN Pipeline-CNN fue una computadora paralela con 64 procesadores de los cuales solo 32 fueron exclusivos para las pruebas de este trabajo; se contó con una memoria principal de 8 GB con una arquitectura de memoria compartida distribuida y buses de alta velocidad. Con respecto a los resultados de clasificación para el CPAN Pipeline-CNN entrenado con la base de datos de los cuatro tipos de virus de la hepatitis C, se obtuvo una precisión del 95% con 145 imágenes probadas y al final de la etapa 4000 la precisión en el entrenamiento fue del 94.5% y la precisión de la validación fue del 95 % (ver tabla II). Al usar el CPAN Pipeline-CNN con las clases EI e IE, se obtuvo una precisión de evaluación de 80.8% con 177 imágenes de prueba y al final de la etapa 4000 la precisión de entrenamiento fue de 82% y la precisión de validación fue de 75% (ver tabla III). Los resultados del entrenamiento del CPAN Pipeline-CNN donde se usaron las tres clases de la base de datos muestran una precisión de evaluación del 57,5% con 301 imágenes y al final de la etapa 4000 la precisión de entrenamiento fue del 69% y la precisión de validación fue del 56% (ver tabla IV).

Tabla II. Precisión de entrenamiento y validación del CPAN Pipeline-CNN con clases de virus de hepatitis C tipo 1, 2, 3 y 6.

Pasos de Entrenamiento (145 imágenes procesadas)				
	1000 pasos	*2000 pasos*	*3000 pasos*	*4000 pasos*
Precisión de Entrenamiento	92%	95%	96%	94.5%
Precisión de Validación	91%	92%	95%	96%

Tabla III. Precisión de entrenamiento y validación del CPAN Pipeline-CNN con clases EI e IE

Pasos de Entrenamiento (177 imágenes procesadas)				
	1000 pasos	*2000 pasos*	*3000 pasos*	*4000 pasos*
Precisión de Entrenamiento	77%	80%	81.7	82%
Precisión de Validación	73%	74.7%	75%	75%

Tabla IV. Precisión de entrenamiento y validación del CPAN Pipeline-CNN con clases EI, IE y N

Pasos de Entrenamiento (301 imágenes procesadas)				
	1000 pasos	*2000 pasos*	*3000 pasos*	*4000 pasos*
Precisión de Entrenamiento	57%	60%	64%	69%
Precisión de Validación	52%	56%	55%	56%

Con respecto al rendimiento de CPAN Pipeline-CNN para el caso de estudio que se ha mostrado, se llevó a cabo el análisis de aceleración obteniendo buenos resultados según el modelo de CPAN. Los gráficos de las fig. 11, 12, 13 y 14 muestran dicho análisis de rendimiento con valores que van desde los 1000 pasos de entrenamiento hasta los 4000 pasos. En dichos gráficos se muestra la aceleración de la precisión de entrenamiento y de validación del CPAN Pipeline-CNN con clases de virus de Hepatitis C tipo 1, 2, 3 y 6, clases de IE y EI y clases de IE, EI y N respectivamente. En todos ellos, las medidas obtenidas del speedup muestran una aceleración al incorporar en cada análisis más CPU-SET, siempre por debajo de la ley de Amdahl. Los tiempos de ejecución en cada entrenamiento varían: Para el caso de 1000 pasos de entrenamiento, de un tiempo de ejecución secuencial promedio de 24 minutos, se obtuvo una disminución con 32 CPU-SET de 11.3 minutos promedio. Para el caso de 2000 pasos de entrenamiento, de un tiempo de ejecución secuencial promedio de 28 minutos, se obtuvo una disminución con 32 CPU-SET de 17 minutos promedio. En el caso de los 3000 pasos de entrenamiento, se obtuvo un tiempo de ejecución secuencial promedio de 33 minutos, mientras que la ejecución paralela con 32 CPU-SET fue de 14.8 minutos promedio. Finalmente, para el caso de 4000 pasos de entrenamiento, el tiempo de ejecución secuencial promedio fue de 40 minutos y la ejecución paralela con 32 CPU-SET lo redujo en un promedio de 20.1 minutos.

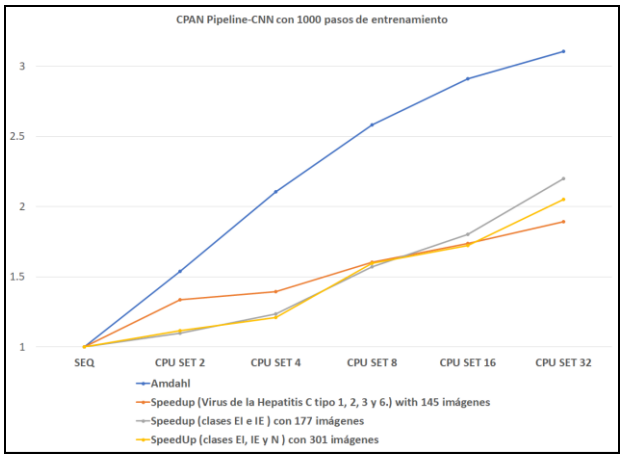


Fig 11. Escalabilidad de Speedup encontrada para el CPAN Pipeline-CNN respecto de precisión de entrenamiento y validación con 1000 pasos de entrenamiento

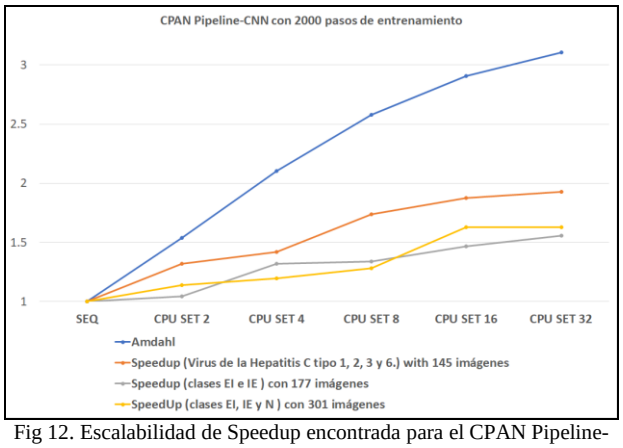


Fig 12. Escalabilidad de Speedup encontrada para el CPAN Pipeline-CNN respecto de precisión de entrenamiento y validación con 2000 pasos de entrenamiento

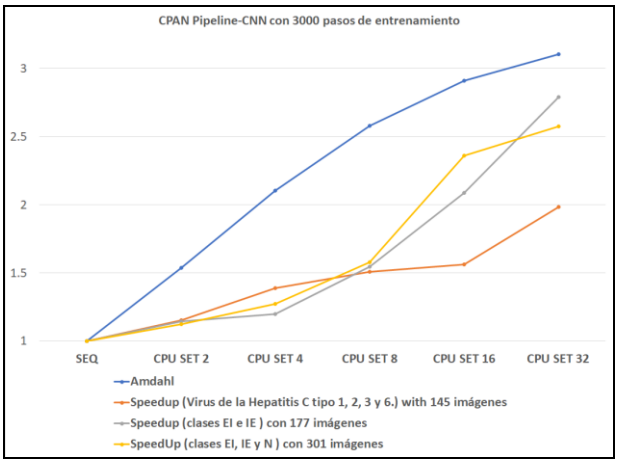


Fig 13. Escalabilidad de Speedup encontrada para el CPAN Pipeline-CNN respecto de precisión de entrenamiento y validación con 3000 pasos de entrenamiento

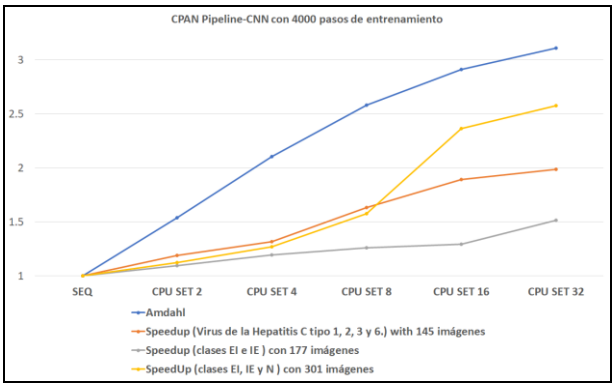


Fig 14. Escalabilidad de Speedup encontrada para el CPAN Pipeline-CNN respecto de precisión de entrenamiento y validación con 4000 pasos de entrenamiento

VII. Conclusiones

Se mostró la implementación del CPAN Pipeline-CNN como un patron de comunicación/interacción abstracto y reutilizable entre procesos que implementa una red neuronal convolucional (CNN) con transferencia de aprendizaje profundo, utilizando un pipeline como patrón de comunicación asociado. Este CPAN puede incluso ser utilizado por programadores novel de aplicaciones paralelas y obtener un código eficiente programando solo las partes secuenciales de sus aplicaciones (objetos esclavos del modelo de la fig. 7). Para demostrar la utilidad del CPAN propuesto, éste fue entrenado en el reconocimiento de secuencias de ADN mediante representaciones gráficas y poder obtener así, una clasificación de los diferentes tipos de virus de la hepatitis C (tipo 1, 2, 3 y 6). Los resultados obtenidos del CPAN Pipeline-CNN entrenado con la base de datos del virus de la hepatitis C sugieren que la metodología de aprendizaje automático utilizada en este trabajo es la adecuada para la clasificación de imágenes generadas a partir de secuencias de ADN. Se muestran buenos porcentajes de precisión de evaluación, precisión de entrenamiento y precisión de validación. La transferencia de aprendizaje es buena cuando hay pocas imágenes disponibles para entrenar al CPAN Pipeline-CNN y permite alcanzar resultados aceptables en la mayoría de los casos (esto se puede ver en las tablas II, III y IV), aunque los resultados pueden mejorarse. Por otro lado, la ejecución paralela del CPAN Pipeline-CNN muestra un buen rendimiento al comparar su aceleración con respecto a su ejecución secuencial. También hemos obtenido un buen rendimiento en la velocidad de sus ejecuciones y una escalabilidad del speedup en comparación con la ley de Amdahl en cuanto al número de procesadores utilizados en el entrenamiento (ver fig. 11, 12, 13 y 14).

Referencias

[1] Calvo D., Red Neuronal Convolucional (CNN). Data Scientist. http://www.diegocalvo.es/red-neuronal-convolucional/. 2015.

[2] Vizcaya R., Deep Learning para la detección de peatones y vehículos sobre FPGA. Disertación de Master. Universidad Autónoma del Estado de México. 2018.

[3] Brinch Hansen, Model Programs for Computational Science: A programming methodology for multicomputers. Concurrency: Practice and Experience. Volume 5, Number 5, 1993.

[4] Corradi A., Leonardi L., PO Constraints as tools to synchronize active objects. Journal Object Oriented Programming 10, pp. 42-53, 1991.

[5] Wilkinson B., Allen M., Parallel Programming Techniques and Applications Using Networked Workstations and Parallel Computers. Prentice-Hall. USA. 1999.

[6] Roosta, Séller, Parallel Processing and Parallel Algorithms. Theory and Computation. Springer. 1999.

[7] Rossainz, M., Capel M., A Parallel Programming Methodology using Communication Patterns named CPANS or Composition of Parallel Object. 20TH European Modeling & Simulation Symposium.Campora S. Giovanni. Italy. 2008.

[8] Rossainz M., Capel M., Approach class library of high-level parallel compositions to implements communication patterns using structured parallel programming. 26TH European Modeling & Simulation Symposium. Bordeaux, France. 2014.

[9] Rossainz, M., Una Metodología de Programación Basada en Composiciones Paralelas de Alto Nivel (HLPCs). Universidad de Granada, PhD dissertation, 2005.

[10] Corradi A, Leonardo L, Zambonelli F., Experiences toward an Object-Oriented Approach to Structured Parallel Programming. DEIS technical report no. DEIS-LIA-95-007. 1995.

[11] Bacci, Danelutto, Pelagatti, Vaneschi, SklE: A Heterogeneous Environment for HPC Applications. Parallel Computing 25, 1999.

[12] Danelutto M. and Torquati M, Loop parallelism: a new skeleton perspective on data parallel patterns, in Proc. of Intl. Euromicro

PDP. Parallel Distributed and Network-based Processing, Torino, Italy. 2014.
[13] Birrell A., An Introduction to Programming with Threads. Digital Equipment Corporation, Systems Research Center. Palo Alto California, USA 1989.
[14] Lavander G.R., Kafura D.G., A Polimorphic Future and First-class Function Type for Concurrent Object-Oriented Programming. Journal of Object-Oriented Systems. 1995.
[15] Andrews G.R., Foundations of Multithreaded, Parallel, and Distributed Programming. Addison-Wesley 2000.
[16] Liwu Li, Java Data Structures and Programming. Springer Verlag. Germany. ISBN: 3-540-63763X. 2002.
[17] Salzberg SL, Searls DB, and Kasif S., Computational gene prediction using neural networks and similarity search. Computational Methods in Molecular Biology, pp.32-109. 1998.
[18] Mathworks, Deep learning. https://la.mathworks.com/solutions/deeplearning/convolutional-neural-network.html. 2018.
[19] Marcelo A., Apolloni J., Kavka C., et-al., Entrenamiento de Redes Neuronales. Universidad Nacional de San Luís. WICC 2000. Argentina. 2000.
[20] Marturet R., Alferez E.S., Evaluación de Redes Neuronales Convulcionales para la clasificación de imágenes histológicas de cancer colorrectar mediante transferencia de aprendizaje. Master en Bioinformática y Bioestadística. Universitat Oberta de Catalunya. España. 2018.
[21] Christos Ouzounis, Rise and demise of bioinformatics? promise and progress. PLoS computational biology, 8(4):e1002487. 2012.
[22] Panduro A, Biología molecular en la clínica. McGraw-Hill Interamericana. 2009.
[23] Noordewier M, Towell G, and Shavlik Jude, Training knowledge-based neural networks to recognize genes in DNA sequences. In Advances in neural information processing systems, pages 530–536. 1991.
[24] Genís P, Blanco P. and Guigó R., Geneid in drosophila. Genome research, 10(4):511–515. 2000.

Redes Neuronales Convolucionales para el Modelado del Rendimiento del Producto Matriz–Vector Disperso

Maria Barreda,[1] Manuel F. Dolz,[1] M. Asunción Castaño,[1]
Pedro Alonso-Jordá[2] y Enrique S. Quintana-Ortí[2]

Resumen— **Modelar el tiempo de ejecución del producto matriz-vector disperso (SpMV) en una arquitectura CPU actual es una tarea compleja debido a:** *i)* **los accesos de memoria irregulares,** *ii)* **referencias indirectas a memoria, y** *iii)* **su baja intensidad aritmética. Si bien los modelos analíticos pueden proporcionar estimaciones precisas para el número total de aciertos/fallos de caché, a menudo no pueden predecir con precisión el tiempo de ejecución. En este trabajo se utilizan redes neuronales convolucionales (RNCs) como herramientas alternativas para estimar el rendimiento del SpMV. Para ello, se diseña una representación de la matriz dispersa basada en bloques de elementos distintos de cero que sirve como entrada para las RNCs. Los resultados experimentales en un subconjunto de matrices dispersas de la colección SuiteSparse Matrix demuestran que los modelos RNCs son lo suficientemente robustos para predecir el rendimiento de la operación SpMV en un núcleo de un procesador Intel Xeon Haswell.**

Palabras clave— **Producto Matriz-Vector Disperso (SpMV), Modelado del Rendimiento, Entrenamiento Supervisado, Redes Neuronales Convolucionales (RNCs).**

I. Introducción

El producto matriz-vector disperso (SpMV) es una operación de suma importancia en numerosas aplicaciones científicas y de ingeniería [1,2]. En muchas de estas aplicaciones, el SpMV es una de las operaciones que más tiempo consume, debido en parte a la limitación por la velocidad de acceso a memoria que sufre. En este sentido, estimar el tiempo de ejecución del SpMV es una tarea compleja debido a: *i)* los accesos de memoria irregulares, *ii)* las referencias indirectas a memoria, y *iii)* su baja intensidad aritmética. Algunos elementos clave que dictan su rendimiento son el patrón de dispersión y la densidad por fila de los elementos no nulos de la matriz dispersa y que, junto con el algoritmo, determinan la secuencia de accesos a memoria y los fallos de caché. En los últimos años, se han desarrollado numerosos modelos de rendimiento del SpMV [3]. Sin embargo, los modelos analíticos propuestos solo proporcionan estimaciones teóricas de los accesos a memoria y operaciones aritméticas, lo que requiere un conocimiento exhaustivo de la arquitectura de CPU y un análisis detallado de la implementación del SpMV [4].

[1]Departamento de Ingeniería y Ciencia de los Computadores, Universitat Jaume I, 12.071–Castellón, {mvaya,dolzm,castano}@uji.es

[2]Departamento de Sistemas Informáticos y Computación, Departamento de Informática de Sistemas y Computadores, Universitat Politècnica de València, 46022–Valencia, {palonso@upv.es,quintana@disca.upv.es}

El aprendizaje automático se ha postulado como un enfoque alternativo a estos modelos analíticos que permite derivar modelos matemáticos mediante datos de entrenamiento, a través del aprendizaje supervisado. Las redes neuronales (RNs), en particular, tienen la capacidad de aproximar comportamientos no lineales a la vez que son adaptativos, es decir, pueden aprender a partir de nuevos datos. Por esta razón, las RNs pueden considerarse como una técnica efectiva para estimar el rendimiento de algoritmos complejos que dependen de accesos de memoria irregulares, como es el caso del SpMV. Y concretamente las RNCs son capaces de capturar dependencias espaciales y temporales mediante representaciones de alto nivel de la matriz dispersa.

Este trabajo utiliza las RNCs como herramientas para identificar los patrones de dispersion y características de la matriz dispersa involucrada en la operación SpMV con el objetivo de proporcionar una estimación precisa del tiempo de ejecución en un núcleo de una CPU moderna. En concreto, este trabajo realiza las siguientes aportaciones:

- Se utilizan las RNCs para modelar el tiempo de ejecución del SpMV en un núcleo de un procesador Intel Xeon Haswell utilizando el formato de almacenamiento CSR [5].
- Se propone una abstracción por bloques de la matriz dispersa para hacer que la arquitectura del modelo de RNC sea independiente de las dimensiones de la matriz dispersa, así como para aumentar la cantidad de datos de entrenamiento/validación.
- Se evalúa la precisión y se demuestra la robustez de los modelos basados en RNCs utilizando un subconjunto representativo de casos que surgen de aplicaciones reales presentes en la colección SuiteSparse Matrix.

El resto del artículo se estructura de la siguiente forma. La sección II revisa los conceptos básicos sobre la operación SpMV y las RNCs. La sección III describe la estrategia para adaptar el formato CSR como una entrada válida para las RNCs y detalla las arquitecturas de la red propuesta. La sección IV evalúa el proceso de entrenamiento, el ajuste de los hiperparámetros y analiza la precisión alcanzada por las redes. La sección V resume algunos trabajos relacionados. Finalmente, la sección VI concluye el trabajo y enumera trabajos futuros.

II. Antecedentes

A. Producto matriz-vector disperso

Considérese la operación SpMV como $y = Ax$, donde A es una matriz dispersa de tamaño $m \times n$, que contiene nnz elementos no nulos, x es un vector de entrada denso, de tamaño n, e y es el vector de salida denso, de tamaño m. En esta operación, los elementos de la matriz A se almacenan habitualmente usando un formato comprimido, como *Compressed Sparse Row* (CSR), *Compressed Sparse Column* (CSR), *Coordinate* (COO) o *Ellpack* (ELL) [5]. En este trabajo nos centramos en el formato CSR, ya que ofrece una solución flexible, eficiente en memoria e independiente de la arquitectura.

El formato CSR almacena la matriz usando tres vectores que contienen los valores no nulos, el principio/fin de cada fila, y el índice de columna de cada elemento, haciendo un uso eficiente de la memoria y permitiendo un acceso rápido a las filas. La Figura 1 proporciona un ejemplo sencillo de una matriz 4×4 almacenada en este formato. Entonces:

- El vector *vval*, de tamaño *nnz*, almacena los elementos no nulos de A.
- En el vector *vptr*, de longitud $n+1$, la diferencia entre los elementos $i+1$ e i especifica el número de elementos no nulos en la fila i-ésima de A. ($2 - 0 = 2$ en la primera fila, $3 - 2 = 1$ en la segunda fila, etc.)
- Las entradas del vector *vpos* especifican el índice de columna de cada entrada de la matriz A, y por tanto, también tiene tamaño *nnz*. (Nótese que el ejemplo asume un indexado que empieza por 0.)

$$\begin{pmatrix} 1 & 0 & 0 & 2 \\ 0 & 0 & 3 & 0 \\ 0 & 4 & 5 & 0 \\ 0 & 0 & 0 & 6 \end{pmatrix} \quad \begin{aligned} vval[6] &= \{1, 2, 3, 4, 5, 6\} \\ vptr[5] &= \{0, 2, 3, 5, 6\} \\ vpos[6] &= \{0, 3, 2, 1, 2, 3\} \end{aligned}$$

Fig. 1: Ejemplo de una matriz almacenada en formato CSR.

El Algoritmo 1 muestra la implementación de la operación SpMV con la matriz A almacenada en formato CSR. El bucle externo (indexado por i) itera sobre las filas de la matriz, mientras que el bucle interno (indexado por j) se mueve a través de las entradas de cada fila, usando los vectores *vptr* y *vpos* para recuperar el índice adecuado que accede a x. Finalmente, *vval* se utiliza para obtener el valor de la coordenada (i, j) de A.

Algoritmo 1 Implementación de la operación SpMV utilizando el formato CSR.

```
Require: A → m × n, x → n, y → m
for i = 1, 2, ..., n do
    for j = vptr[i], vptr[i] + 1, ..., vptr[i + 1] − 1 do
        y[i] := y[i] + vval[j] · x[vpos[j]]
    end for
end for
```

B. Redes neuronales convolucionales

Las RNCs son una clase de redes neuronales muy eficientes para identificar patrones en problemas de clasificación de datos [6]. En general, una RNC de L capas consiste en una colección de C capas convolucionales (CONV), normalmente dispuestas en las primeras capas de la red, seguidas de un reducido número $F = L - C$ de capas completamente conectadas (CC) en las etapas finales de la red. Las neuronas en una capa l de tipo CONV están conectadas a un pequeño subconjunto de neuronas de la capa $l - 1$, y se activan de acuerdo al resultado de la operación convolucional usando un filtro de n dimensiones. Una capa CONV puede combinar múltiples filtros, cada uno responsable de detectar una característica compleja y no lineal que produce un único mapa de características en la capa l.

Para reducir la dimensión de los datos de entrada de la RNC a medida que se procesan a través de la red se colocan capas de *pooling* entre las sucesivas capas CONV. El objetivo de estas capas es reducir progresivamente el tamaño espacial y disminuir el número de parámetros y el cálculo en la red. Una operación de *pooling* típica es la operación *max*.

Las RNCs también pueden conterner capas de *dropout*, que se insertan para mejorar el proceso de entrenamiento y evitar el sobreentrenamiento de la red (*overfitting*). La idea que subyace en este tipo de capas es ignorar las neuronas con baja probabilidad para ayudar a que la red aprenda características más robustas, lo que a su vez conduce a una mejor generalización ante nuevos datos de entrada. Las RNCs también pueden incluir capas de optimización, conocidas como *batch normalization* [7]. El propósito de estas capas es normalizar las activaciones de una capa en cada lote para que la activación media y la desviación estándar estén cerca de 0 y 1, respectivamente.

Una vez que todos los mapas de características se han procesado a través del conjunto de C capas CONV, el conjunto de F capas CC genera un resultado para la RNC. Una capa CC l conecta cada una de sus neuronas con todas las neuronas en la capa $l - 1$, siguiendo los mismos principios que los de los perceptrones multicapa (PMs) tradicionales. Dependiendo de si la RNC modela un problema de clasificación o uno de regresión, la capa l puede contener tantas neuronas como clases, o una sola. En el primer caso, las neuronas se activan a través de funciones no lineales (por ejemplo, sigmoide, softmax o ReLU), mientras que en el último, la única neurona de salida puede activarse a través de una función lineal.

III. Modelado del SpMV mediante RNCs

En esta sección se presenta la metodología utilizada para estimar el rendimiento del SpMV utilizando RNCs. La naturaleza de la operación SpMV, limitada por memoria, se debe a las bajas densidades de los elementos distintos de cero y a los patrones irregulares de dispersión en la matriz A que, en ge-

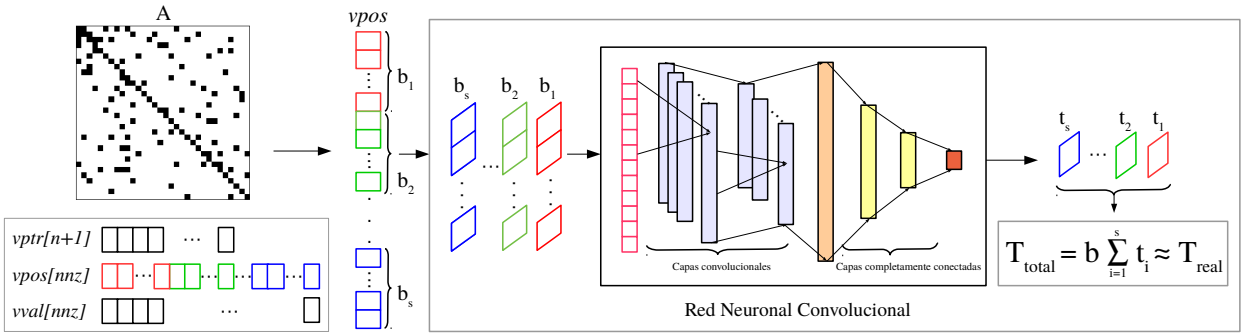


Fig. 2: Flujo de trabajo para el modelado del rendimiento de SpMV.

neral, generan un volumen considerable de fallos de caché y accesos a la memoria DRAM para recuperar las entradas del vector x. Teniendo esto en cuenta, el vector *vpos* puede considerarse como un elemento clave para comprender las distintas intensidades de acceso a memoria y predecir el tiempo de ejecución global de la operación. Con esta idea, se propone una RNC donde las entradas son los valores del vector *vpos* del formato CSR. En este sentido, el vector puede verse como una imagen unidimensional de la matriz dispersa A que captura el orden en que las entradas de x se recuperan de la memoria y las distancias entre accesos consecutivos a este vector. Una vez entrenados, los filtros en las capas convolucionales deberían ser capaces de capturar características significativas en el vector *vpos* que produzcan estimaciones útiles del rendimiento de SpMV a través de la relación entre los flops y los aciertos/fallos de caché.

A. Metodología

La Figura 2 muestra la metodología propuesta para abordar el problema de modelado del SpMV. Como se ha mencionado, el objetivo es diseñar una RNC que reciba el vector *vpos* como entrada. Sin embargo, dado que las matrices dispersas pueden presentar grandes variaciones en su tamaño y número de elementos no nulos (nnz), se propone dividir el vector *vpos* en bloques de tamaño b para obtener un diseño de RNC fijo con un número constante de entradas. La ventaja de este enfoque es que las RNCs se pueden usar de manera uniforme para predecir los tiempos de ejecución de bloques de igual tamaño, que pueden pertenecer a cualquier matriz dispersa, independientemente de su tamaño y valor de nnz. Por lo tanto, considerando que t_i es el tiempo de ejecución por elemento no nulo del bloque i-ésimo de A, el tiempo total de ejecución para esta matriz se puede calcular a partir de la suma del tiempo por elemento para los $\lceil nnz/b \rceil = s$ bloques, multiplicado por b; es decir, $T_{total} \approx b \sum_{i=1}^{s} t_i$. Nuestra metodología proporciona iterativamente a la RNC los bloques del vector *vpos* de modo que las salidas proporcionadas por la red se corresponden con las estimaciones del tiempo de ejecución parcial (por elemento no nulo) de cada bloque. Finalmente, los resultados parciales se acumulan para obtener el tiempo total de ejecución del SpMV asociado a la matriz A.

Algoritmo 2 Implementación de la operación SpMV por bloques utilizando el formato CSR.

Require: $A \rightarrow m \times n, x \rightarrow n, y \rightarrow m, rows \rightarrow nnz, b \rightarrow block\_size$
1: **for** $i = 0, 1, \ldots, m-1$ **do**
2: **for** $j = vptr[i], vptr[i]+1, \ldots, vptr[i+1]-1$ **do**
3: $rows[j] := i$
4: **end for**
5: **end for**
6: $start := 0$
7: $end := start + b$
8: **while** $start < nnz$ **do**
9: $prv := m$
10: $aux := 0{,}0$
11: *start_timer()*
12: **for** $i = start, start+1, \ldots, end-1$ **do**
13: **if** $rows[i] > prv$ **then**
14: $y[rows[i-1]] := y[rows[i-1]] + aux$
15: $aux := 0{,}0$
16: **end if**
17: $aux := aux + vval[i] \cdot x[vpos[i]]$
18: $prv := rows[i]$
19: **end for**
20: *stop_timer()*
21: $y[prv] := y[prv] + aux$
22: $start := end$
23: **if** $(start + b) < nnz$ **then**
24: $end := start + b$
25: **else**
26: $end := nnz$
27: **end if**
28: **end while**

El particionado de *vpos* en bloques, sin embargo, nos obliga a implementar una versión por bloques del algoritmo clásico del SpMV basado en CSR (Algoritmo 1). Esta modificación es necesaria para generar el conjunto de datos de entrenamiento para las RNCs, ya que cada bloque de *vpos* tiene que etiquetarse con su correspondiente tiempo de ejecución por elemento no nulo. El Algoritmo 2 presenta el procedimiento para calcular el SpMV por bloques de b elementos no nulos. Para cada bloque, el algoritmo calcula el producto de los elementos distintos de cero (de acuerco con *vpos*) por los valores correspondientes del vector de entrada x. El resultado se almacena en la posicion análoga del vector de salida y (líneas 14 y 21). Cabe destacar que las líneas 11 y 20 incluyen las instrucciones para la medición el tiempo de cada bloque para etiquetarlo y poder entrenar la RNC.

B. Arquitecturas de red neuronal

Teniendo en cuenta la estrategia por bloques propuesta, el próximo paso es diseñar una arquitectura para la RNC que ofrezca estimaciones precisas del tiempo de ejecución para SpMV. Con este propósito, tratamos la tarea de modelado como un problema de clasificación. Básicamente, una RNC de este tipo tiene n neuronas de salida (clases) en la última capa que especifican la probabilidad de que una muestra de entrada pertenezca a cierta clase. En nuestro caso, las clases de salida corresponden a intervalos de tiempo del mismo tamaño que juntas incluyen todo el rango de t_{nnz} en el conjunto de datos. Por lo tanto, la activación de un cierto intervalo de tiempo de salida en la RNC de clasificación significa que el t_{nnz} predicho para un cierto bloque de entrada está en ese intervalo. Con este enfoque, para la estimación del t_{nnz} de dicho bloque se toma el centroide de dicho intervalo.

Para diseñar la arquitectura de la red, nos hemos inspirado en la estructura de algunas RNCs de vanguardia (p.e., AlexNet, LeNet o VGG-16), en las que la tendencia común es apilar bloques de capas convolucionales combinadas con capas de *pooling* al final (véase la Sección II-B). Concretamente, optamos por una secuencia de una o dos capas convolucionales, seguidas de una capa de *pooling* y repetimos esta secuencia dos veces. Estos dos modelos se denotan como Conv-Pool y Conv-Conv-Pool, respectivamente. Después de las capas de *pooling* se añaden capas de *dropout* y de normalización. En la segunda parte de la red se incluyen capas CC que combinan las características detectadas en las primeras capas convolucionales. Finalente, la última capa CC genera la salida para en el modelo de clasificación. Siguiendo las ideas previas, se han diseñado dos arquitecturas de RNC que combinan las dos disposiciones diferentes de capas convolucionales. En concreto, el modelo que sigue el patrón Conv-Pool lo denotamos como RNC-C1, mientras que el que utiliza el patrón Conv-Conv-Pool, lo denominamos RNC-C2 en el artículo.

IV. Evaluación experimental

En este apartado, se describe *i)* la generación de los conjuntos de datos de aprendizaje y evaluación utilizados en la experimentación, *ii)* la construcción de los modelos de RNCs, *iii)* el proceso de aprendizaje, y *iv)* la evaluación de las redes entrenadas sobre el conjunto de matrices dispersas de prueba en función de los errores medios relativos del tiempo de ejecución de la operación SpMV. Para llevar a cabo estas tareas hemos empleado los siguientes componentes hardware y software:

- *Hardware:*
  - Las redes se han entrenado en dos procesadores Intel Xeon E5-2698, con un total de 40 núcleos a 2,20 GHz y cuatro GPU NVIDIA Tesla P100 con 16 GB de DRAM a 1,48 GHz interconectados a través de NVLink.
  - Los tiempos de ejecución correspondientes a la operación SpMV se han obtenido en un núcleo Intel Xeon E5-2630 a 2,40 GHz, al que en adelante denominaremos Haswell.
- *Software:* El entorno utilizado para construir y entrenar las RNs es Keras v2.2.4 [8], que se ejecuta sobre TensorFlow r1.10 [9]. Además, se emplea Hyperas v0.4.1 [10], un software de optimización de hiperparámetros para Keras. El algoritmo de la operación SpMV se ha implementado en C y se ha compilado con GCC 5.3.0.

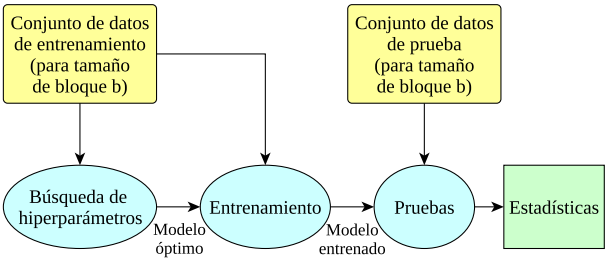


Fig. 3: Flujo de trabajo de la evaluación.

En la Figura 3 se muestra el flujo de trabajo seguido para el entrenamiento y evaluación de los modelos. En primer lugar, se construyen los conjuntos de datos de entrenamiento y prueba para un tamaño de bloque dado, b. A continuación, se obtienen las versiones optimizadas de los modelos a entrenar buscando los hiperparámetros más adecuados para estos. Después, se entrenan y se prueban los modelos utilizando respectivamente los conjuntos de datos mencionados anteriormente. Finalmente, se acumula el tiempo de ejecución que el modelo entrenado ha estimado para cada bloque de una misma matriz dispersa con el fin de obtener el tiempo de ejecución total de la operación SpMV estimado para cada matriz del conjunto de datos de prueba. A partir de estas estimaciones se calculan los errores relativos con respecto a los tiempos de ejecución reales.

A. Conjuntos de entrenamiento, validación y prueba

Los conjuntos de datos de entrenamiento y prueba se han obtenido ejecutando la operación SpMV por bloques (como se detalla en el Algoritmo 2) para las matrices dispersas seleccionadas y midiendo el tiempo por elemento no nulos, t_{nnz}, para cada uno de los bloques *vpos* de las matrices. Para este propósito, se han seleccionado 173 matrices dispersas de la colección de matrices SuiteSparse [11] con un número de elementos no nulos que oscilan entre 1 M y 10 M. Del total de matrices, 108 (63 %) se han seleccionado para el entrenamiento, mientras que las 65 restantes (37 %) se han reservado para probar los modelos entrenados. De manera similar, el 80 % del conjunto de datos de entrenamiento se ha empleado para el entrenamiento en sí y el 20 % restante, para la validación del modelo, con el fin de guiar el proceso de entrenamiento de este y de evitar el sobrentrenamiento.

Para analizar el impacto del tamaño del bloque también experimentamos con diferentes valores de $b \in \{250, 500, 750, 1000, 3000, 5000\}$. Con ese

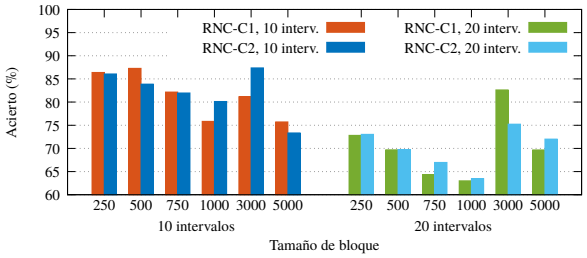


Fig. 4: Porcentaje de muestras de validación bien reconocidas con las RNCs en función del número de intervalos y del tamaño de bloque.

propósito, se han obtenido diferentes conjuntos de datos de entrenamiento, validación y prueba para cada valor de b.

B. Construcción de los modelos

Para cada uno de los tamaños de bloque seleccionados se ha implementado el modelo correspondiente utilizando Keras. Además, se han considerado dos variantes de los modelos que emplean 10 y 20 intervalos de salida para el rango t_{nnz}. Antes de iniciar la fase de entrenamiento es necesario configurar un conjunto de hiperparámetros relacionados con la topología de los modelos y con el entrenamiento en sí. Entre estos hiperparámetros se encuentran el número de filtros en las capas CONV, las dimensiones estos filtros, el número de capas CC y el número de neuronas en cada una, el tamaño del lote, el factor de aprendizaje, los porcentajes de *dropout* y el algoritmo de optimización del entrenamiento. Ahora bien, probar y establecer hiperparámetros manualmente es un proceso engorroso y propenso a errores, además de inviable por el enorme espacio de búsqueda que hay que analizar en un tiempo razonable. Keras dispone de una herramienta, denominada Hyperas, que resuelve el problema de la estimación de hiperparámetros de forma subóptima utilizando algoritmos de búsqueda bayesianos [12].

C. Entrenamiento

El objetivo de las RNCs propuestas es maximizar el porcentaje de muestras correctamente clasificadas. Para lograrlo, el entrenamiento se lleva a cabo partiendo de los valores obtenidos en el proceso de búsqueda de hiperparámteros para cada situación.

La Figura 4 muestra los porcentajes de muestras de validación correctamente clasificadas con los modelos RNC-C1 y RNC-C2 usando 10 y 20 clases y teniendo en cuenta los diferentes tamaños de bloque considerados. En primer lugar se observa que las RNCs con 10 clases obtienen generalmente mejores resultados. Esto es debido a que el tamaño del intervalo de tiempo asociado a una de las 10 clases es mayor que el tamaño correspondiente para 20 clases. Nótese también que el ratio de acierto es mucho mayor con menos intervalos. Sin embargo, usar menos intervalos implica obtener en general errores más altos en la fase de prueba, ya que el tiempo estimado t_{nnz} que se adopta es el centroide del intervalo. En otras palabras, el máximo error relativo (η) cometido en la estimación dentro del intervalo $(a, b]$ es $\eta = \left|\frac{(a+b)/2-b}{b}\right| = \left|\frac{a-b}{2b}\right|$. Una segunda observación sobre la Figura 4 es que los modelos RNC-C1 y RNC-C2 no tienen siempre igual comportamiento para un mismo tamaño de bloque, de manera que, según dicho tamaño, se obtienen mejores resultados con el modelo Conv-Pool o con Conv-Conv-Pool. Finalmente se observa que los modelos que usan 10 intervalos obtienen mejores tasas de clasificación con bloques pequeños, excepto para $b = 3000$.

D. Evaluación

Una vez entrenados los modelos, el siguiente paso es evaluarlos utilizando el conjunto de datos de prueba, compuesto de 65 matrices dispersas diferentes a las usadas en el entrenamiento. La métrica de evaluación utilizada para ello en esta ocasión es el Error Medio Relativo (EMR), que mide el error relativo promedio entre el tiempo de ejecución total estimado y medido para todas las matrices de prueba, es decir, $EMR = \frac{1}{p}\sum_{i=1}^{p}\frac{|estimado_i - medido_i|}{medido_i}$, donde p es el número total de matrices en el conjunto de datos de prueba.

La figura 5 muestra el EMR obtenido sobre el conjunto de datos de prueba para los modelos en los diferentes tamaños de bloque seleccionados. Esta métrica varía de forma general entre 1 % y 7 %, lo que revela que los modelos proporcionan estimaciones bastante buenas. En general, se detecta que aquellos modelos que utilizan la configuración Conv-Conv-Pool suelen generar un EMR más bajo que aquellos que usan configuración Conv-Pool. Además, el uso de 10 intervalos en lugar de 20 no proporciona un EMR más bajo. También se ha visto que un tamaño de bloque pequeño genera un EMR más bajo.

A continuación seleccionamos aquellas RNCs con las que se obtuvieron los mejores resultados en el análisis anterior y analizamos el Error Relativo (ER) obtenido con ellas para algunas de las matrices de prueba. Los resultados pueden verse en Figura 6. Las matrices se ordenan ascendentemente en función del ER obtenido con el modelo RNC-C1. Los resultados muestran que el ER es inferior al 1 % para aproximadamente la mitad de las matrices de prueba en las dos configuraciones.

V. Estado de la cuestión

Las RNCs son herramientas de aprendizaje supervisado que se remontan a los años 60 y 70, aunque el aprendizaje profundo ha adquirido mucha popularidad recientemente tras la aparición de arquitecturas aceleradoras y del procesamiento de datos masivos. El éxito de las RNCs radica, por un lado, en su habilidad de extraer de forma automática durante el proceso de entrenamiento características de los datos a procesar y, por otro, en su capacidad de manejar grandes conjuntos de datos. Sin embargo, aunque las RNCs se han aplicado con éxito en muchas áreas relacionadas con el aprendizaje automático [13, 14] prácticamente no se han explorado aún en el campo

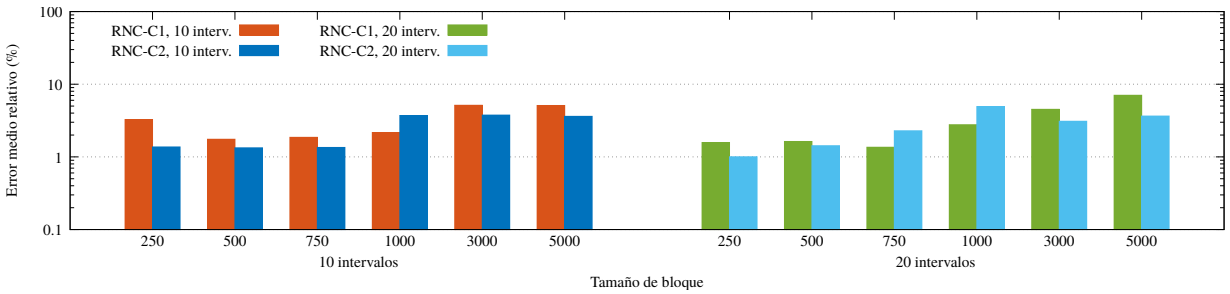


Fig. 5: Error medio relativo en el tiempo de ejecución de las RNCs.

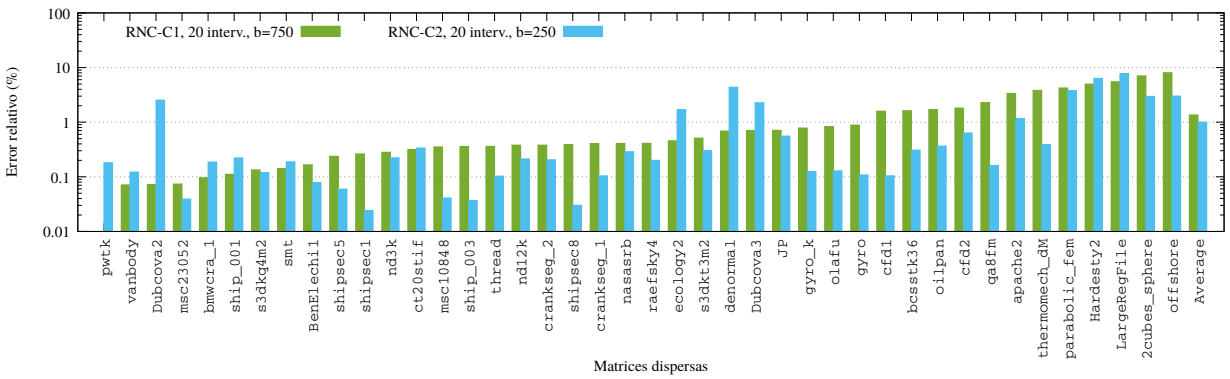


Fig. 6: Error relativo en el tiempo de ejecución de las mejores RNC-C1 y RNC-C2 para las matrices de prueba.

del álgebra lineal. Concretamente, solo unos pocos trabajos, que se describen brevemente a continuación, han abordado problemas relacionados con el modelado del rendimiento y la operación SpMV utilizando el aprendizaje profundo.

Götz y Anzt [15] plantean el patrón de dispersión de una matriz como una imagen que proporcionan a una RNC que entrenan para detectar los bloques de elementos no nulos de la matriz y obtener los precondicionadores de Jacobi. De manera similar, Zhao et al. [16] emplean RNCs para seleccionar el formato más adecuado para almacenar la matriz dispersa involucrada en una operación SpMV. En lugar de usar la matriz implicada en la operación SpMV como una imágen 2D, Nisa et al. [17] proporcionan a un PM diferentes características de las matrices dispersas procesadas para predecir el mejor formato de almacenamiento de dichas matrices.

Con respecto al modelado del rendimiento, Tiwari et al. [18] emplean PMs para estimar el rendimiento, la potencia y el uso de energía de algunos núcleos computacionales a partir de ciertas características del núcleo a resolver y de la matriz a abordar. Benatia el al. [19] entrenan PMs a partir de una serie de características del formato de la matriz a procesar para predecir el rendimiento de la GPU en la operación SpMV usando diferentes tipos de formatos de almacenamiento de las matrices.

VI. Conclusiones y trabajos futuros

En este trabajo se han diseñado varias RNCs basadas en clasificación que permiten estimar el tiempo de ejecución del SpMV, una operación con importantes aplicaciones en problemas científicos y de ingeniería. Las RNCs capturan los patrones complejos y las características de la matriz dispersa (almacenada en formato CSR), que básicamente dicta los accesos irregulares al vector denso de entrada. Para conseguir que la arquitectura de las RNCs sea independiente del tamaño de la matriz dispersa se ha empleado una estrategia por bloques. Para optimizar la arquitectura de los modelos de RNCs, sus hiperparámetros se han ajustado mediante algoritmos de búsqueda bayesianos.

Las redes propuestas se han entrenado con un conjunto de matrices dispersas de la colección de matrices SuiteSparse etiquetadas con el tiempo de ejecución correspondiente a la operación SpMV en un núcleo de Haswell. El EMR para el conjunto de matrices de prueba para todos los modelos propuestos varía entre 1 % y 7 %. Estos resultados muestran que las arquitecturas de red analizadas ofrecen resultados precisos, sobre todo para bloques pequeños. Además, la experimentación revelea que incrementar los intervalos de salida en las RNCs no mejora el EMR, aunque reduce la precisión general del modelo.

Como trabajo futuro, se plantea extender las RNCs para estimar el tiempo de ejecución y el consumo de energía de la implementación paralela de la operación SpMV. Un objetivo adicional es aprovechar esta metodología para modelar el tiempo de ejecución y el consumo de energía de otras operaciones de álgebra lineal más complejas.

Agradecimientos

Este trabajo ha sido financiado por el proyecto TIN2017-82972-R del *MINECO*. Manuel F. Dolz ha recibido financiación del Plan GenT CDEIGENT/2018/014 de la *Generalitat Valenciana* y Maria Barreda la ha recibido de la ayuda postdoctoral POSDOC-A/2017/11 de la *Universitat Jaume I*.

Referencias

[1] Ahmad Abdelfattah, Hatem Ltaief, and David Keyes, "High performance multi-gpu spmv for multi-component pde-based applications," in *Euro-Par 2015: Parallel Processing*, Jesper Larsson Träff, Sascha Hunold, and Francesco Versaci, Eds., Berlin, Heidelberg, 2015, pp. 601–612, Springer Berlin Heidelberg.

[2] William E Schiesser, *Computational mathematics in engineering and applied science: ODEs, DAEs, and PDEs*, CRC press, 2014.

[3] Ping Guo and Liqiang Wang, "Accurate cross–architecture performance modeling for sparse matrix–vector multiplication (spmv) on gpus," *Concurrency and Computation: Practice and Experience*, vol. 27, no. 13, pp. 3281–3294, 2015.

[4] K. Li, W. Yang, and K. Li, "Performance analysis and optimization for spmv on gpu using probabilistic modeling," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 1, pp. 196–205, Jan 2015.

[5] Victor Eijkhout and Roldan Pozo, "Data structures and algorithms for distributed sparse matrix operations," Tech. Rep., 1994.

[6] Jiuxiang Gu, Zhenhua Wang, Jason Kuen, Lianyang Ma, Amir Shahroudy, Bing Shuai, Ting Liu, Xingxing Wang, Gang Wang, Jianfei Cai, and Tsuhan Chen, "Recent advances in convolutional neural networks," *Pattern Recognition*, vol. 77, no. C, pp. 354–377, May 2018.

[7] Sergey Ioffe and Christian Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," 2015, pp. 448–456.

[8] "Keras: The Python Deep Learning library," `https://keras.io/`.

[9] "TensorFlow, an open source machine learning library for research and production.," `https://www.tensorflow.org/`.

[10] "Keras + Hyperopt: A very simple wrapper for convenient hyperparameter optimization," `http://maxpumperla.com/hyperas/`.

[11] "SuiteSparse Matrix Collection," `https://sparse.tamu.edu/`.

[12] J. Bergstra, D. Yamins, and D. D. Cox, "Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures," in *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*. 2013, ICML'13, pp. I–115–I–123, JMLR.org.

[13] Jürgen Schmidhuber, "Deep learning in neural networks: An overview," *Neural Networks*, vol. 61, pp. 85 – 117, 2015.

[14] Yann LeCun, Y Bengio, and Geoffrey Hinton, "Deep learning," *Nature*, vol. 521, pp. 436–44, 05 2015.

[15] Markus Götz and Hartwig Anzt, "Machine learning-aided numerical linear algebra: Convolutional neural networks for the efficient preconditioner generation," in *Procs of ScalA'18: 9th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems, WS at Supercomputing 2018*, 11 2018.

[16] Yue Zhao, Jiajia Li, Chunhua Liao, and Xipeng Shen, "Bridging the gap between deep learning and sparse matrix format selection," *SIGPLAN Not.*, vol. 53, no. 1, pp. 94–108, February 2018.

[17] Israt Nisa, Charles Siegel, Aravind Sukumaran Rajam, Abhinav Vishnu, and P Sadayappan, "Effective machine learning based format selection and performance modeling for spmv on gpus," EasyChair Preprint no. 388, EasyChair, 2018.

[18] A. Tiwari, M. A. Laurenzano, L. Carrington, and A. Snavely, "Modeling power and energy usage of hpc kernels," in *2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops PhD Forum*, May 2012, pp. 990–998.

[19] A. Benatia, W. Ji, Y. Wang, and F. Shi, "Machine learning approach for the predicting performance of spmv on gpu," in *2016 IEEE 22nd International Conference on Parallel and Distributed Systems (ICPADS)*, Dec 2016, pp. 894–901.

Simulador para la gestión de recursos de cómputo: evaluación de distintos métodos de selección

César Gómez-Martín [1] y Miguel A. Vega-Rodríguez [2]

Resumen— **En una sociedad consciente de la importancia del ahorro energético la correcta gestión de recursos de cómputo es una tarea muy importante para reducir la huella de carbono debido al aumento de la demanda computacional. Por ello, la simulación de entornos de cómputo utilizando nuevos algoritmos de gestión de recursos puede ser de gran ayuda para los analistas de sistemas, gestores de centros de datos o de infraestructuras TIC. Existen varios simuladores que calculan el rendimiento y, de alguna forma, también estiman el consumo energético, pero no hay ninguno en el que el modelo energético esté basado en datos que hayan sido validados por organismos independientes como la Standard Performance Evaluation Corporation (SPEC). Este es el motivo por el que hemos desarrollado un simulador de entornos HPC denominado Performance and Energy Aware Scheduling (PEAS).**

Para evaluar el simulador hemos propuesto un nuevo algoritmo basado en NSGA-II, un algoritmo genético multiobjetivo, para la selección de recursos de cómputo. Con el simulador PEAS hemos podido demostrar que es posible usar políticas de asignación de recursos para ahorrar energía y tiempo de cómputo sin compromenter el rendimiento.

Los resultados de nuestras simulaciones muestran grandes mejoras en el tiempo de respuesta y reducción del consumo energético. En la mayoría de casos nuestra implementación de NSGA-II tiene un mejor desempeño que otros algoritmos "inteligentes" como MOHEFT, y siempre supera al clásico algoritmo de asignación de recursos first-fit.

Podemos concluir por lo tanto que el simulador es muy útil para este tipo de estudios ya que hemos demostrado que los algoritmos multiobjetivo mejoran el rendimiento de los algoritmos normalmente usados por los gestores de recursos de HPC actuales.

Palabras clave— **soptimización multiobjetivo, algoritmos de scheduling, HPC, simulador, evaluación de rendimiento**

I. Introducción

Hoy en día, con la llegada de dispositivos empotrados, los portátiles, los móviles, y también, por los altos costes energéticos y el calentamiento global, se hace necesaria una gestión eficiente de la energía que consumen los dispositivos electrónicos. Esta eficiencia es ya una prioridad para los fabricantes de hardware y para los gestores de centros de datos. En el pasado, y durante más de medio siglo, la comunidad científica dedicada al cómputo en general, y a la supercomputación en particular, se ha dedicado casi en exclusiva al incremento del rendimiento de los sistemas y ha dejado atrás otros aspectos importantes como la eficiencia energética. Ahora, en esta sociedad más consciente del malgasto energético la planificación de tareas de cómputo se ha convertido en un tema importante para los ingenieros y para los analistas de sistemas, y el ahorro energético lleva a éstos a investigar fórmulas que permiten aumentar el rendimiento por vatio de las infraestructuras.

Los planificadores de tareas actuales son una pieza clave en los supercomputadores o clústeres de cómputo moderno porque, si los configuramos correctamente, podemos reducir los costes de operación de forma drástica.

En la literatura hay múltiples artículos que estudian el impacto del escalado dinámico de frecuencia y del voltaje en los procesadores (DVFS), combinándolo con un algoritmo para ahorrar energía [1], y también hay otros trabajos sobre el balanceo y el ahorro de energía producido por el uso de máquinas virtuales [2].

Una de las innovaciones que presentamos en este artículo es la inclusión del benchmark "Power and Performance" de la Standard Performance Evaluation Corporation (SPEC). Este benchmark lo usamos para el modelado de consumo energético y rendimiento de un clúster de servidores o supercomputadores en los que se ejecutan diferentes tareas mediante el uso de un planificador de tareas. Además, nuestro simulador incluye la implementación del algoritmo NSGA-II [3], un algoritmo genético multiobjetivo rápido y elitista. Con él vamos a estudiar si es posible realizar un emplazamiento inteligente de los trabajos de cómputo que sea capaz de ahorrar tiempo y energía sin comprometer el rendimiento del sistema.

A. Trabajos relacionados

Los fabricantes e integradores de hardware han intentado siempre desarrollar herramientas para una mejor gestión de entornos HPC. De hecho, existen algunas soluciones comerciales y otras libres para la gestión de diferentes cargas de trabajo:

- SLURM es un gestor de recursos de código abierto que se ha convertido en el más popular entre los mayores supercomputadores del mundo por su versatilidad para la gestión de clústeres heterogeneos de cualquier tamaño [4].
- IBM Platform LSF (Load Sharing Facility) es una plataforma de gestión de trabajos de cómputo de gran demanda que está basada en el proyecto Utopía de la Universidad de Toronto

[1]Universidad de Extremadura - Escuela Politécnica de Cáceres, Avda. Universidad s/n - 10.003 Cáceres (España), e-mails: cesar@unex.es.

[2]Universidad de Extremadura - Escuela Politécnica de Cáceres, Avda. Universidad s/n - 10.003 Cáceres (España), e-mails: mavega@unex.es.

[5].
- Univa Grid Engine es un gestor de recursos distribuido que amplía las capacidades de Open Grid Engine [6].
- TORQUE (Terascale Open-source Resource and QUEue manager) es una extensión de PBS (Portable Batch System) que desarrolló la NASA para asignar recursos a tareas de cómputo [7].

Según [8], si diseñamos un sistema de gestión de trabajos de cómputo tenemos que tener en cuenta los tipos de datos siguientes:

- Datos de usuario: se utilizan para determinar las prioridades de los trabajos de cómputo.
- Datos de petición de recursos: estos datos especifican el número de procesadores que se necesitan, su arquitectura, cantidad de memoria, una estimación del tiempo de ejecución, etc.
- Datos de los objetivos de planificación: son los datos que ayudan al planificador a generar "buenos" planes.

La mayoría de los autores están de acuerdo en que un planificador debe tener tres componentes software bien diferenciados: la política de programación de trabajos, la función objetivo y los algoritmos de planificación. Sin embargo otros investigadores introducen un cuarto elemento decisorio al que denominan "política de selección de recursos" [9]. Este cuarto elemento se usa para asignar los recursos en función de las limitaciones o de los requerimientos del trabajo. Hay simuladores como MuPSiE [10] que siguen la filosofía de usar tres componentes y otros que utilizan cuatro como Alvio [9] y Kento-sim [11]. Otros investigadores proponen la gestión energética de entornos multiprocesador usando técnicas de machine learning como en [12], pero hay una carencia de simuladores que combinen rendimiento, gestión de energía y de recursos con algoritmos de optimización multiobjetivo.

B. Contribuciones

En este artículo presentamos nuestro simulador PEAS, pensado para ayudar en la toma de decisiones para una correcta configuración de los sistemas de planificación de recursos de cómputo en centros de cálculo. Adicionalmente, y con la ayuda del simulador, pretendemos desarrollar nuevos algoritmos de selección de recursos y de planificación de tareas para mejorar el rendimiento y el consumo energético. El simulador es capaz de recrear cargas de trabajo reales de los mayores supercomputadores del mundo y también puede utilizar políticas de selección de recursos y planificación que ya existen. Estas políticas serán comparadas con nuestro algoritmo multiobjetivo basado en NSGA-II [3].

En el resto del artículo describiremos brevemente la arquitectura del simulador PEAS en la Sección II. A continuación, en la Sección III explicaremos cómo se procesan los trabajos dentro del simulador para calcular el rendimiento, el consumo energético y métricas temporales. En la Sección IV presentaremos nuestra implementación de NSGA-II y en la Sección V mostraremos comparativas entre NSGA-II y otros algoritmos que están aceptados por la industria. A posteriori, en la Sección VI esbozaremos qué trabajos futuros tenemos en mente y posibles mejoras para el simulador. Por último, en la Sección VII explicaremos las conclusiones a las que hemos llegado después de realizar este trabajo.

II. Arquitectura del simulador

El simulador PEAS está concebido para evaluar si los recursos que se utilizan en entornos de cómputo son los óptimos. Dispone de un planificador que está dirigido por eventos, es decir, el simulador no es consciente del próximo evento hasta que este ocurre (llegada de un nuevo trabajo, finalización de un trabajo, etc.). El planificador y el selector de recursos funcionan mediante una secuencia discreta de eventos espaciados en el tiempo, cada evento acontece en un momento concreto y eso significa que hay un cambio en el sistema. El código fuente del simulador PEAS se puede descargar de la siguiente dirección: https://github.com/cesargomez/peas. En ese repositorio también se puede descargar un ejemplo de fichero de configuración, varios ficheros con clústeres de ejemplo y algunas trazas en formato SWF [13]. En las subsecciones siguientes se explicará cada uno de los componentes de forma detallada.

A. Modelado de la carga de trabajos

En los centros de cómputo modernos hay normalmente un software encargado de realizar la gestión de los trabajos que se envían a los clústeres y de guardar datos estadísticos sobre los mismos. Estos datos son muy útiles a la hora de estudiar las diferentes políticas y algoritmos para la gestión y planifiación de recursos. Existen trazas de ejecución que han sido generadas de forma aleatoria o benchmarks sintéticos que podrían servir para hacer estudios teóricos pero que nunca van a llegar a la precisión que dan las trazas de trabajos de cómputo que han sido enviados a supercomputadores reales. Es por esto por lo que en el simulador PEAS utilizamos trazas que han sido descargadas del Parallel Workload Archive [14] [15], que se trata de un repositorio con trazas obtenidas en diferentes supercomputadores de distinta naturaleza. Para ello utilizamos el formato de traza SWF que fue propuesto por David Talby y refinado posteriormente por Dror Feitelson, James Patton Jones y otros [13].

B. Modelado de rendimiento y consumo energético

El Intel Thermal Design Power (TDP) es la máxima potencia que un procesador puede usar en un periodo significativo. Por contra, AMD Average CPU Power (ACP) que es la media geométrica de la temperatura que un procesador puede disipar al ejecutar cuatro benchmarks diferentes TPC Benchmark*-C, SPECcpu, SPECjbb and STREAM. Tanto Intel como AMD están de acuerdo en el hecho de que no es buena idea medir el consumo energético

sumando el consumo de cada componente según indicado en sus especificaciones puesto que eso sólo indica el consumo máximo. En el caso que nos ocupa, y dependiendo del tipo de trabajo de cómputo, puede haber muchos elementos que estén ociosos la mayor parte del tiempo por lo que la mejor manera de hacer los cálculos es utilizando un medidor de corriente. Además, la potencia en sí misma no es una medida de eficiencia, tiene que haber un compromiso entre el rendimiento y el consumo, es decir, debemos tener en cuenta el rendimiento por cada vatio porque un sistema que consuma poco pero tenga bajo rendimiento puede tardar más tiempo en hacer una determinada tarea y puede que llegue a consumir más energía. Para evitar este tipo de problemas, SPEC diseñó el benchmark "SPEC Power and Performance", que es el primer estándar de la insdustria para evaluar las características de consumo y rendimiento de clústeres multinodo [16]. Nuestro simulador PEAS tiene un componente que es capaz de analizar los ficheros proporcionados por SPEC en formato CSV y asignar esos datos a cada servidor que utilicemos.

En las tablas I y II hemos incluido dos ejemplos de ficheros de configuración de clústeres que muestran los campos más relevantes de los mismos.

C. Componente de planificación de tareas

Otro de los elementos importantes de nuestro sistema es el componente que denominamos Scheduling Policy que es el encargado de almacenar los trabajos en una estructura de tipo lista que está ordenada por tiempo de llegada. Los algoritmos de planificación más comunes son los basados en "first-come first-served" (FCFS) porque son fáciles de comprender desde el punto de vista del usuario y también porque generan planificaciones más o menos equitativas. El algoritmo FCFS tradicional es muy ineficiente y solamente se usa para hacer estudios o experimentos, en su lugar, los sistemas de planificación modernos utilizan las variantes con relleno ("backfilling") que permiten que trabajos con una menor prioridad puedan usar los recursos libres.

El simulador tiene las siguientes variantes implementadas:

- FCFS: que es muy ineficiente debido a la enorme fragmentación que produce, pero a la vez es muy útil porque es muy sencillo de entender de forma teórica. Si el trabajo más prioritario tiene recursos suficientes puede comenzar su ejecución, si no los tiene éste y el resto de trabajos encolados deben esperar a que haya recursos suficientes.
- FCFS Conservative Backfilling: esta variante adelanta los trabajos con menor prioridad, es decir, aquellos que llegaron más tarde, solamente si ninguno de los jobs con más prioridad se retrasan. Esta variante no afecta a los trabajos que ya están encolados, por eso es denominado como la versión conservadora [17].
- FCFS EASY Backfilling: fue desarrollado por IBM para su supercomputador SP2, también se conoce como "backfilling" agresivo porque permite que los trabajos cortos se ejecuten antes sólo si no retrasan al primer trabajo de la cola. El resto de trabajos sí podrían verse afectados [18].

D. Componente de asignación de recursos

Este componente es el que nos permite implementar algoritmos de optimización multiobjetivo, es el encargado de decidir en qué nodos y procesadores se van a ejecutar los trabajos de cómputo. Dependiendo de las limitaciones en el número de procesadores, memoria, o consumo puede que seleccione un servidor diferente si es necesario. La política de selección más simple que viene normalmente definida en los gestores de recursos es "first-fit", con esta política el sistema intenta encontrar las primeras unidades de ejecución que cumplen los requisitos del trabajo y las reserva. Dado que el tiempo de ejecución y el consumo energético son dos parámetros importantes en este artículo y, dado que ambos parámetros son igual de importantes, proponemos la implementación de un algoritmo evolutivo multiobjetivo (MOEA) para dotar de eficiencia a este componente. Hay una gran variedad de MOEAs pero hemos descubierto que NSGA-II [3] consigue muy buenos resultados cuando hay que minimizar dos parámetros que son igual de importantes. Otro de los motivos para decantarnos por NSGA-II es por su velocidad, es muy importante que un gestor de recursos sea lo más rápido posible en la toma de decisiones, existen otros algoritmos evolutivos que retrasarían esa toma de decisiones y esa situación no sería ideal [19]. Desafortunadamente no existen muchos algoritmos en la literatura que tengan en cuenta el consumo energético per se, pero hemos encontrado que MOHEFT [20] es una buena opción para realizar comparativas con nuestro algoritmo.

III. Metodología

Los servidores de cómputo tienen un rendimiento dispar, la mayoría de los benchmarks, especialmente los que se basan en medir el desempeño en un periodo de tiempo concreto, son claramente engañosos puesto que las mediciones se realizan en el momento de máxima utilización del sistema. Por ello, es importante utilizar benchmarks que puedan cuantificar la relación entre la capacidad de cómputo junto con el consumo energético, por eso usamos el benchmark SPEC Power and Performance [16] en nuestro simulador.

Los parámetros que el simulador calcula son los siguientes:

- Tiempo de ejecución (runtime): se calcula para cada trabajo de la lista y depende fundamentalmente de las características de los nodos de cómputo en los que se ejecuta.
- Consumo energético: también se calcula para cada trabajo. Por ejemplo, si el trabajo usa solamente la mitad de un servidor su consumo será la mitad de consumo total de ese servidor. Ya que un trabajo puede ejecutarse en varios servidores su consumo será la suma de los consumos

TABLA I
PRIMER EJEMPLO DE FICHERO DE CONFIGURACIÓN DE CLÚSTER

OEM	Procesador	#Chips	#Cores/Chip	RAM (GB)	OPS@100%Carga	Vatios@100%Carga
ASUS	Xeon L5420	2	4	8.00	270621	170
ASUS	Xeon L5430	2	4	8.00	278927	173
Acer	Xeon X3470	1	4	8.00	309401	125
Acer	Xeon X5670	2	6	12.0	898544	267
Acer	Xeon X5670	2	6	12.0	897310	265
Acer	Xeon X5670	2	6	12.0	890144	272
ASUS	Xeon X3360	1	4	4.00	165064	118
Acer	Xeon X3470	1	4	8.00	309401	125
Acer	Xeon E3-1260L	1	4	8.00	295443	58
Acer	Xeon X3470	1	4	8.00	308318	124
Media	-	1.5	4.6	8.8	373302.9	169.7

TABLA II
SEGUNDO EJEMPLO DE FICHERO DE CONFIGURACIÓN DE CLUSTER

OEM	Procesador	#Chips	#Cores/Chip	RAM (GB)	OPS@100%Carga	Vatios@100%Carga
Fujitsu	Xeon L5430	2	4	8.00	326264	221
Fujitsu	Xeon L5430	2	4	8.00	293162	220
Bull	Xeon X5670	2	6	12.0	911120	247
HP	Xeon L5430	2	4	16.0	306620	253
HP	Xeon 5160	2	2	8.00	159151	258
HP	Xeon L5530	2	4	8.00	534763	187
SuperMicro	Xeon E5345	2	4	16.0	245571	334
HP	Xeon X5670	2	6	16.0	902465	249
HP	Xeon L5640	2	6	16.0	725620	172
HP	Xeon X5675	2	6	16.0	894314	222
Media	-	2	4.6	12.4	529905	236.3

de todas las porciones usadas de cada servidor.
- Rendimiento: es una métrica muy importante puesto que depende de otros factores, es decir, el tiempo de ejecución, la carga del sistema y el algoritmo de selección de recursos.

En las subsecciones siguientes se detallarán las diferentes métricas utilizadas para realizar los cálculos.

A. Métricas de los tiempos de ejecución

La mayoría de algoritmos de planificación confían en las estimaciones proporcionadas por los usuarios para realizar la asignación de recursos, sin embargo, en nuestro simulador realizamos el cálculo del número de operaciones por segundo (OPS) correspondientes a cada trabajo para tener una medida lo más fidedigna posible. Para ello se asume que el tiempo real del trabajo de cómputo que se ha obtenido del fichero SWF es el tiempo que tardaría en ejecutarse en un "procesador promedio". El procesador promedio se calcula de la siguiente forma:

$$OPS_{CPU_{Promedio}} = \frac{\sum_{i=1}^{nc} OPS_{CPU_i}}{nc} \quad (1)$$

donde:

- OPS_{CPU_i} es el número de operaciones por segundo del procesador con una carga del 100%
- nc es el número total de CPUs del cluster.

Con el tiempo de ejecución del trabajo podemos calcular el tiempo de ese mismo trabajo en otro procesador de la siguiente manera:

$$RuntimeSimulado_{Trabajo} = \left(\max_{i=1...CPUsAsignadas} \frac{OPS_{CPU_{Promedio}}}{OPS_{CPU_i}}\right) * Runtime_{Trabajo} \quad (2)$$

donde:

- $Runtime$ es el tiempo de ejecución real del trabajo en el listado de la traza.
- OPS_{CPU_i} es el número de operaciones por segundo al 100% de carga en CPU_i.
- $OPS_{CPU_{Promedio}}$ es el número de operaciones por segundo al 100% de carga en $CPU_{Promedio}$ (Ver 1).
- $CPUsAsignadas$ es el número de CPUs en las que se ejecuta el trabajo.

B. Métricas de consumo energético

Debido a las medidas de reducción del gasto, y probablemente también al calentamiento global, la reducción del coste energético de los trabajos de cómputo es una tarea en la que los gestores e ingenieros están poniendo mucho empeño.

Como hemos indicado en secciones anteriores, nuestro simulador utiliza un modelo basado en el benchmark SPECpower [21] porque es el primer benchmark energético aceptado por la industria.

Para entender los siguientes cálculos es necesario tener en cuenta lo siguiente; a pesar de que existen muchos servidores y sistemas cuyo rango de uti-

lización se encuentra entre el 5% y el 20% de uso la mayoría de supercomputadores y centros de cómputo operan cercanos al 95%-100% de capacidad. Los sistemas de cómputo suelen estar diseñados para arrojar el mejor ratio rendimiento/consumo energético cuando operan al 100% [22] (no es este el caso de otros dispositivos que usan baterías). En la mayoría de los casos es más eficiente ejecutar a la frecuencia y voltajes máximos por un corto periodo de tiempo y después mantener el servidor en un estado ocioso por más tiempo, a esto se denomina "race to idle" [23]. Esta es la razón por la que en muchos centros de cómputo el escalado dinámico de frecuencia y de voltaje no se tienen en cuenta [24]. Por esto, en nuestro simulador calculamos la energía necesaria para ejecutar un trabajo de la siguiente manera:

$$Consumo_{Trabajo} = \sum_{i=1}^{CPUsAsignadas} Vatios@100\%Carga_{CPU_i} * RuntimeSimulado_{Trabajo} \quad (3)$$

donde:

- $Vatios@100\%Carga_{CPU_i}$ es el consumo energético de CPU_i al 100% de carga. (Ver tablas I y II).
- $CPUsAsignadas$ es el número de CPUs en las que se ejecuta el trabajo.

C. Métricas de rendimiento

Existen algunos malentendidos que es conveniente aclarar, ya que a veces se asume que un servidor de bajo consumo es siempre más eficiente. El consumo per se no es una medida de eficiencia si no se combina con el rendimiento. Un sistema con bajo consumo y menor rendimiento necesita más tiempo para ejecutar una tarea y puede que eso le lleve a consumir más energía. Los servidores más eficientes son aquellos que tienen el mejor rendimiento por vatio. El simulador PEAS está pensado para comparar estrategias y algoritmos que proporcionan el mejor ratio rendimiento/consumo, pero no debemos olvidarnos de los usuarios, ya que una de las tareas de los gestores de recursos y de los planificadores es dotar a los sistemas de un entorno sensible al tiempo de espera de los usuarios, es por ello por lo que el simulador también calcula otras métricas centradas en los usuarios [25] que pasamos a explicar:

- Tiempo medio de espera del trabajo o "average waiting time" (AWT):

$$AWT = \frac{\sum_{i=1}^{n} t_i^w}{n} \quad (4)$$

donde:
- t_i^w es el tiempo de espera del $Trabajo_i$.
- n es el número total de trabajos aceptados por el planificador.

- Tiempo medio de respuesta o "average response time" (ART):

$$ART = \frac{\sum_{i=1}^{n} t_i^r}{n} \quad (5)$$

donde:
- t_i^r es el tiempo de respuesta del $Trabajo_i$.
- n es el número total de trabajos aceptados por el planificador.

- Tiempo medio de ejecución o "average execution time" (AET):

$$AET = \frac{\sum_{i=1}^{n} t_i^e}{n} \quad (6)$$

donde:
- t_i^e es el tiempo de ejecución del $Trabajo_i$.
- n es el número total de trabajos aceptados por el planificador.

IV. Una nueva política de asignación de recursos basada en NSGA-II

Otro de los objetivos de nuestro artículo es la comparación de nuestro algoritmo NSGA-II con el algoritmo first-fit y MOHEFT, un algoritmo reciente que tiene en cuenta el rendimiento y el consumo energético (MOHEFT [20]).

- First-fit es un algoritmo al que no le importa si una solución es buena o no, siempre va a devolver la primera solución que cumpla con los requisitos del trabajo. Este algoritmo revisa la matriz de procesadores del clúster para encontrar los n primeros que satisfacen los requisitos. Se trata del algoritmo más utilizado por los gestores de recursos actuales y suele venir configurado por defecto.
- El algoritmo MOHEFT mejora y amplía HEFT, el algoritmo HEFT (Heterogeneous Earliest Finish Time) [26] es uno de los más populares para planificar trabajos de cómputo. MOHEFT es una versión multiobjetivo que optimiza dos objetivos: tiempo de ejecución y consumo energético. Los resultados que se obtienen con este algoritmo son mejores que los de otros algoritmos similares por esto lo hemos escogido para hacer comparativas con nuestra algoritmo basado en NSGA-II.
- NSGA-II [3] es un algoritmo evolutivo multiobjetivo que puede usarse para la optimización de objetivos que son contradictorios entre sí como el tiempo de ejecución y el consumo energético (los procesadores que consumen menos tardan más tiempo en ejecutar por norma general).

A. Optimización multiobjetivo con NSGA-II

Como se ha explicado en secciones anteriores, en este artículo intentamos proponer mejoras para la planificación de tareas de cómputo de manera que

tengamos en cuenta el consumo energético de las mismas, pero a la vez, sin que la reducción de ese consumo suponga una degradación importante en el rendimiento. En el caso que nos ocupa queremos optimizar dos variables de decisión que se corresponden con los mínimos de dos funciones objetivos [27] y que son contradictorias entre sí, es decir, que una mejora en una de ellas suele causar el empeoramiento de la otra. Podemos encontrar similares problemas en muchas disciplinas científicas y resolverlos no ha sido tarea sencilla para los investigadores.

Encontrar una solución multiobjetivo perfecta es casi imposible, de ahí la complejidad y la belleza de este tipo de técnicas. Una solución razonable sería encontrar un conjunto de soluciones en el que cada una de ellas satisfaga los objetivos de forma aceptable sin ser dominada por ninguna otra solución [28].

Las características principales de NSGA-II son:

- Método de ordenación en el que cada individuo (o solución) es ordenado de acuerdo a un nivel de no-dominación.
- Elitismo que permite almacenar todas las soluciones no dominadas para mejorar las propiedades de convergencia.
- Implementación de una comparativa básada en la distancia de hacinamiento (crowding distance) para garantizar la diversidad y propagar las soluciones.
- Implementación de restricciones usando la definición de dominancia sin usar funciones de penalización; hay muchos métodos en la literatura que usan técnicas reparadoras, técnicas de penalización, de separación o incluso técnicas híbridas que castigan la velocidad del algoritmo [29].

Como se puede ver en el pseudocódigo de nuestra implementación en Algorithm 1, al comienzo inicializamos la población de padres (P_0) de forma aleatoria. En cada iteración se combinan los padres y los hijos después de que hayan sido ordenados en función de sus niveles de Pareto utilizando el algoritmo de ordenación "fast non-dominated sort".

Para una población de tamaño N, un procedimiento de ordenación lento necesitaría que cada solución fuese comparada con el resto de soluciones para dilucidar su dominancia (complejidad $O(MN)$ donde M es el número de objetivos). Cuando este proceso finalice sólo tendríamos la primera solución del primer frente de Pareto no dominado. Para calcular el resto de soluciones del primer frente la complejidad total sería $O(MN^2)$, además, para hacer lo mismo con el siguiente frente no dominado deberíamos repetir el proceso para el peor caso ($O(MN^3)$).

En definitiva, el algoritmo "fast non-dominated sort" nos permite reducir la complejidad a $O(MN^2)$ [3] y obtener las soluciones de una forma más rápida.

V. Resultados

En esta sección realizamos la comparativa de first-fit, MOHEFT y NSGA-II utilizando el simulador PEAS.

A. Selección del algoritmo de planificación

Para encontrar una buena política de planificación hemos realizado varios experimentos con diferentes algoritmos para hacer más efectiva nuestra batería de simulaciones. Hemos probado FCFS, FCFS Conservative Backfilling y FCFS EASY Backfilling usando tres ficheros de trabajos diferentes de los siguientes supercomputadores:

- HPC2N: High-Performance Computing Center North (HPC2N) de Suecia.
- LLNL: colección de trabajos de un cluster Linux llamado Thunder que está instalado en el Lawrence Livermore National Lab (LLNL).
- RICC: contiene varios meses de trabajos del supercomputador RIKEN Integrated Cluster of Clusters (RICC).

Todas estas trazas pueden descargarse del Parallel Workload Archive [14]. Los datos de los servidores que se han utilizado en las simulaciones y que se pueden consultar en las tablas I y II están disponibles en la web de SPECpower_ssj2008 [21].

First-fit es el método de selección que viene configurado por defecto en los gestores de recursos es por ello por lo que haremos las comparativas usando este algoritmo.

Los resultados que se han obtenido mediante las simulaciones pueden consultarse en las Tablas III y IV. La mayoría de trabajos de los tres ficheros de trazas son trabajos grandes, es decir, usan muchos recursos, por eso es difícil encontrar el suficiente número de procesadores ociosos para ejecutarlos. La principal razón por la que hay un mejor tiempo de espera media (AWT) usando Conservative Backfilling es porque el planificador garantiza un tiempo de comienzo evitando así la postergación indefinida. En el caso de la traza de LLNL hay un mejor comportamiento con EASY Backfilling porque consta de trabajos más pequeños que son fácilmente planificados en cuanto hay algún hueco libre. Si tenemos en cuenta el tiempo de respuesta medio (ART) para la configuración de la Tabla I, vemos nuevamente que hay un mejor tiempo para el algoritmo Conservative Backfilling para las trazas de HPC2N y RICC porque están compuestas por trabajos de larga duración. Si observamos la Tabla II vemos que hay procesadores menos capaces que los de la otra tabla y por tanto EASY backfilling no los reserva haciendo que los trabajos cortos puedan entrelazarse entre otros trabajos más largos. Vemos que el algoritmo EASY Backfilling beneficia a los trabajos que requieren pocos recursos pero que Conservative Backfilling funciona mejor cuando los trabajos son largos y necesitan muchos recursos. Mientras que Conservative Backfilling proporciona reservas aseguradas EASY Backfilling ofrece mejores oportunidades a trabajos pequeños y

Algorithm 1 Pseudocódigo de NSGA-II

function NSGA-II
  Inicializar el vector de padres $P = \phi$.
  Inicializar el vector de hijos $Q = \phi$.
  Inicializar el vector conjunto $R = \phi$.
  Inicializar el número de generación $t = 0$.
  Asignar valores aleatorios al vector padre P_0.
  while $t < numGenerations$ **do**
    Combinar la población padre con la población de hijos $R_t = P_t \bigcup Q_t$.
    Ordenar soluciones de R_t para obtener todos los frentes no dominados F =fastNonDominatedSort(R_t) donde $F = (F_1, F_2, ...)$.
    Inicializar $P_{t+1} = \phi$ y $i = 1$.
    while tamaño de población de padres $\mid P_{t+1} \mid + \mid F_i \mid < numIndividuals$ **do**
      Añadir el frente no dominado F_i al vector de padres P_{t+1}.
      $i = i + 1$.
    Calcular "crowding distance" de F_i.
    Ordenar F_i en función de la "crowding distance" (crowdingDistanceAssignment(F_i)).
    Rellenar el vector padre P_{t+1} con los primeros $numIndividuals - \mid P_{t+1} \mid$ elementos de F_i.
    Generar la población de hijos Q_{t+1}.
    $t = t + 1$.
P contiene las soluciones no dominadas

TABLA III
MÉTRICAS DE RENDIMIENTO Y CONSUMO DE HPC2N, LLNL Y RICC (1.000 TRABAJOS) - CONFIGURACIÓN DE CLÚSTER DE TABLA I

	FCFS			Conservative			EASY		
Traza	HPC2N	LLNL	RICC	HPC2N	LLNL	RICC	HPC2N	LLNL	RICC
AWT (e+06 seg)	2.43218	2.36677	4.40937	1.67471	2.09732	4.42785	3.46569	1.50004	4.66608
ART (e+06 seg)	2.45016	2.37151	4.57303	1.69166	2.10186	4.5912	3.48084	1.50424	4.8054
AET (seg)	17976	4740	163665	16954	4534	163355	15149	4206	139322
Consumo (KWh)	2072.83	794.807	1744.32	2004.05	786.256	1746.63	1903.12	757.078	1622.68

TABLA IV
MÉTRICAS DE RENDIMIENTO Y CONSUMO DE HPC2N, LLNL Y RICC (1.000 TRABAJOS) - CONFIGURACIÓN DE CLÚSTER DE TABLA II

	FCFS			Conservative			EASY		
Traza	HPC2N	LLNL	RICC	HPC2N	LLNL	RICC	HPC2N	LLNL	RICC
AWT (e+06 seg)	0.22703	2.05046	1.64419	3.6106	1.79806	1.27788	2.27818	1.01472	2.72981
ART (e+06 seg)	0.24306	2.05469	1.76733	3.62618	1.80217	1.40039	2.29356	1.01865	2.8487
AET (seg)	16036	4226	123142	15586	4105	122503	15375	3930	118896
Consumo (KWh)	2298.44	898.178	2653.76	2225.76	887.053	2628.42	2206.59	876.906	2609.74

cortos al haber menos bloqueos en la planificación [30].

Para concluir, dado que nuestras simulaciones arrojan mejores datos de consumo y de tiempo de ejecución en todas las pruebas vamos a utilizar EASY Backfilling para hacer nuestra comparativa entre los algoritmos de gestión de recursos first-fit, MOHEFT y NSGA-II.

B. Resultados experimentales para first-fit, MOHEFT Y NSGA-II

Para comparar las diferentes políticas de selección de recursos tenemos que tener en cuenta que tanto first-fit como MOHEFT son algoritmos deterministas, es decir, siempre producen los mismos resultados. NSGA-II es un algoritmo estocástico por lo que, para dar resultados fiables necesitamos ejecutar varias veces el algoritmo, en nuestro caso vamos a ejecutarlo 10 veces para obtener la media y la desviación estándar.

Es importante enfatizar que las configuraciones que se han utilizado para MOHEFT y NSGA-II son las que los propios autores aconsejan [20] [3]:

- MOHEFT:
  - Tamaño del conjunto de soluciones K: los autores no indican exactamente un valor para K, pero sí dicen lo siguiente [20]: "dado un conjunto de n trabajos y m recursos la complejidad de HEFT es $O(n * m)$ ... Considerando que el conjunto de soluciones es K, el bucle extra de MOHEFT sólo realiza K iteraciones lo que da una complejidad de $O(n * m * K)$. Normalmente el número de soluciones K es un valor mucho menor que n y m... Por tanto la complejidad puede ser cercana a $O(n * m)$, como en el algoritmo HEFT". Por este motivo

TABLA V
First-fit vs. MOHEFT vs. NSGA-II - HPC2N, LLNL y RICC (1.000 trabajos) - Cluster de la tabla I - EASY Backfilling

	First-fit			MOHEFT			NSGA-II		
Traza	HPC2N	LLNL	RICC	HPC2N	LLNL	RICC	HPC2N	LLNL	RICC
AWT Media (e+06 seg)	3.46569	1.50004	4.66608	3.12149	0.8906	3.13486	2.45231	0.209872	4.28811
AWT Desviación est. (e+06)	-	-	-	-	-	-	0.35442	0.067308	1.61927
ART Media (e+06 seg)	3.48084	1.50424	4.8054	3.13552	0.89457	3.26857	2.47018	0.021428	4.36641
ART Desviación est. (e+06)	-	-	-	-	-	-	0.35425	0.067312	1.57394
AET Media (seg)	15149	4206	139322	14028	3965	133711	17876.7	4406.9	78251.8
AET Desviación est.	-	-	-	-	-	-	315.58	60.91	197.59
Consumo medio (KWh)	1903.12	757.078	1622.68	1774.25	730.22	1588.21	2040.43	748.034	1350.456
Desviación est. consumo	-	-	-	-	-	-	12.62	2.88	13.56

TABLA VI
First-fit vs. MOHEFT vs. NSGA-II - HPC2N, LLNL y RICC (1.000 trabajos) - Cluster de la Tabla II - EASY Backfilling

	First-fit			MOHEFT			NSGA-II		
Traza	HPC2N	LLNL	RICC	HPC2N	LLNL	RICC	HPC2N	LLNL	RICC
AWT Media (e+06 seg)	2.27818	1.01472	2.72981	2.48036	0.85492	5.15742	0.013562	0.2497	2.73450
AWT Desviación est.	-	-	-	-	-	-	6128	0.11998e+06	1.40626e+06
ART Media (e+06 seg)	2.29356	1.01865	2.8487	2.49459	0.85854	5.26859	0.023217	0.252578	2.81029
ART Desviación est.	-	-	-	-	-	-	6142	0.119974e+06	1.40644e+06
AET Media	15375	3930	118896	14227	3624	111172	9655.2	2883.3	75793.3
AET Desviación est.	-	-	-	-	-	-	29.99933	24.71052	512.69329
Consumo medio	2206.59	876.906	2609.74	2000.261	750.373	2328.864	872.686	510.697	1494.21
Desviación est. consumo	-	-	-	-	-	-	9.5921	20.98429	13.09628

usamos K=2 (el número de servidores (m) en ambas configuraciones es 10).

- NSGA-II:
  - Número de individuos por generación: 100 individuos o posibles soluciones.
  - Número de generaciones: 100 generaciones o iteraciones.
  - Probabilidad de cruce: 0.9.
  - Probabilidad de mutación: 0.1.

Lo que intentamos optimizar con MOHEFT y NSGA-II es el consumo energético y el tiempo de respuesta medio (ART) de todos los trabajos. En algunas ocasiones puede parecer que el comportamiento de los algoritmos multiobjetivo es peor para ciertos trabajos puesto que el tiempo de ejecución medio (AET) aumenta, pero es muy probable que cuando este tiempo aumenta, el tiempo de respuesta medio (ART) y el consumo estén más optimizados debido a la reducción del tiempo de espera medio (AWT). En todas las simulaciones que hemos realizado se ha optimizado tanto el tiempo de respuesta medio (ART) como el consumo. Después de ejecutar las simulaciones podemos sacar las siguientes conclusiones de los resultados mostrados en las Tablas V y VI:

- Comparativa del tiempo de espera medio (AWT): no se trata de uno de los objetivos que deban ser optimizados por nuestro algoritmo, no obstante se puede observar como en el caso de NSGA-II cuando se optimiza el tiempo de respuesta medio (ART) también hay una mejora substancial de AWT. Esto es debido a que en la mayoría de los casos ART está influido por el tiempo de espera de todos los trabajos de la traza y, por ende, cuando optimizamos ART estamos indirectamente optimizando AWT.
- Comparativa del tiempo de respuesta medio (ART): el tiempo de respuesta medio es mucho mejor cuando se usa NSGA-II. De hecho, en 5 de 6 simulaciones se observa mayor mejora usando NSGA-II que MOHEFT.
- Comparativa del tiempo de ejecución medio (AET): se trata de la métrica que menos mejora al optimizar el tiempo de respuesta medio (ART). Para ejecutar trabajos largos de forma eficiente el algoritmo debe encontrar procesadores con gran rendimiento para optimizar ART, por tanto hay mayor mejora con NSGA-II si los trabajos son largos como se puede observar en los resultados de la traza RICC. MOHEFT es capaz de mejorar el AET de first-fit en todos los casos y obtiene resultados similares a NSGA-II para las trazas HPC2N y LLNL, pero mucho peores resultados si la traza es RICC.
- Comparativa del consumo energético: podemos observar como NSGA-II no solamente disminuye el tiempo de respuesta medio de los trabajos sino también el consumo energético de forma significativa. De hecho, para esta métrica, se obtienen rendimientos muy superiores a first-fit y MOHEFT. Sin embargo, es difícil decir qué soluciones son mejores al comparar MOHEFT y NSGA-II en la Tabla V puesto que las soluciones de ambos son no dominantes entre sí, cuando un algoritmo tiene mejor tiempo de respuesta medio (ART) el otro consigue un menor consumo y viceversa.

Usando NSGA-II obtenemos grandes mejoras de ART y de consumo energético en la mayoría de los casos (nueve de doce), por lo tanto podemos concluir que NSGA-II es superior a first-fit y MOHEFT y por ello recomendamos su uso e implementación en los gestores de recursos.

VI. Trabajo futuro

En el futuro pretendemos mejorar el simulador con las siguientes características:

- El uso de más métricas de rendimiento como las que miden los trabajos en función de su tamaño [25].
- Implementación de otros algoritmos estocásticos, como por ejemplo "scatter search", para comparar si son mejores que la estrategia propuesta en este artículo (NSGA-II).
- Refactorización del mismo para intentar mejorar la velocidad de las simulaciones

VII. Conclusiones

Con la implementación y liberación como código abierto del simulador PEAS intentamos ayudar a aquellos que desean reducir el consumo energético sin menoscabar el rendimiento de sus sistemas de cómputo.

Hoy en día está en boga el uso de inteligencia artigicial, machine learning y la optimización multiobjetivo para realizar una mejor selección de recursos, por ello el simulador ha sido estructurado de tal manera que permita la inclusión de nuevos algoritmos. Del mismo modo que nosotros hemos añadido NSGA-II se pueden añadir otros muchos.

Con la implementación de nuestro algoritmo multiobjetivo de selección de recursos basado en NSGA-II intentamos demostrar que existen mejores algoritmos para la gestión de recursos de HPC que los que normalmente se usan en la actualidad. Es por ello por lo que hemos ejecutado diferentes simulaciones para comparar nuestro enfoque con otros que se usan ampliamente. En todas las ejecuciones hemos comprobado que usando nuestro algoritmo se han optimizado tanto el tiempo de respuesta como el consumo energético de los conjuntos de trabajos ejecutados. De hecho, los resultados muestran una gran mejora en la mayoría de ocasiones (nueve veces de un total de doce ejecuciones) y podemos concluir que NSGA-II da mejor rendimiento que MOHEFT y mucho mejor que first-fit. Por tanto, recomendamos el uso de este algorimo en gestores de recursos modernos como Slurm para ahorrar tiempo y dinero.

Agradecimientos

Este trabajo ha sido parcialmente financiado por la AEI (Agencia Estatal de Investigación, España) y el FEDER (Fondo Europeo de Desarrollo Regional, Unión Europea), bajo el proyecto TIN2016-76259-P (proyecto PROTEIN). Gracias también a la Junta de Extremadura y el FEDER por la ayuda GR18090 otorgada al grupo de investigación TIC015.

Referencias

[1] Kihwan Choi, Ramakrishna Soma, and Massoud Pedram, "Dynamic voltage and frequency scaling based on workload decomposition," in *Proceedings of the 2004 international symposium on Low power electronics and design*. ACM, 2004, pp. 174–179.

[2] Liting Hu, Hai Jin, Xiaofei Liao, Xianjie Xiong, and Haikun Liu, "Magnet: A novel scheduling policy for power reduction in cluster with virtual machines," in *Cluster Computing, 2008 IEEE International Conference on*. IEEE, 2008, pp. 13–22.

[3] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan, "A fast and elitist multiobjective genetic algorithm: Nsga-ii," *Evolutionary Computation, IEEE Transactions on*, vol. 6, no. 2, pp. 182–197, 2002.

[4] Andy B Yoo, Morris A Jette, and Mark Grondona, "Slurm: Simple linux utility for resource management," in *Job Scheduling Strategies for Parallel Processing*. Springer, 2003, pp. 44–60.

[5] Dino Quintero, Luis Carlos Cruz, Ricardo Machado Picone, Dusan Smolej, Daniel de Souza Casali, Gheorghe Tudor, Joanna Wong, et al., *IBM Platform Computing Solutions Reference Architectures and Best Practices*, IBM Redbooks, 2014.

[6] "Univa grid engine software," `http://www.univa.com/products/grid-engine.php`, Acceso: 2019-05-11.

[7] Garrick Staples, "Torque resource manager," in *Proceedings of the 2006 ACM/IEEE conference on Supercomputing*. ACM, 2006, p. 8.

[8] Jochen Krallmann, Uwe Schwiegelshohn, and Ramin Yahyapour, "On the design and evaluation of job scheduling algorithms," in *Job Scheduling Strategies for Parallel Processing*. Springer, 1999, pp. 17–42.

[9] Francesc Guim, Julita Corbalan, and Jesús Labarta, "Modeling the impact of resource sharing in backfilling policies using the alvio simulator," in *Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, 2007. MASCOTS'07. 15th International Symposium on*. IEEE, 2007, pp. 145–150.

[10] Felix Heine, Matthias Hovestadt, Odej Kao, and Achim Streit, "On the impact of reservations from the grid on planning-based resource management," in *Computational Science–ICCS 2005*, pp. 155–162. Springer, 2005.

[11] Francesc Guim, Ivan Rodero, Julita Corbalan, and Manish Parashar, "Enabling gpu and many-core systems in heterogeneous hpc environments using memory considerations," in *High Performance Computing and Communications (HPCC), 2010 12th IEEE International Conference on*. IEEE, 2010, pp. 146–155.

[12] Josep Ll Berral, Íñigo Goiri, Ramón Nou, Ferran Julià, Jordi Guitart, Ricard Gavaldà, and Jordi Torres, "Towards energy-aware scheduling in data centers using machine learning," in *Proceedings of the 1st International Conference on energy-Efficient Computing and Networking*. ACM, 2010, pp. 215–224.

[13] Steve J Chapin, Walfredo Cirne, Dror G Feitelson, James Patton Jones, Scott T Leutenegger, Uwe Schwiegelshohn, Warren Smith, and David Talby, "Benchmarks and standards for the evaluation of parallel job schedulers," in *Job Scheduling Strategies for Parallel Processing*. Springer, 1999, pp. 67–90.

[14] "Parallel workload archive," `http://www.cs.huji.ac.il/labs/parallel/workload/logs.html`, Acceso: 2019-05-11.

[15] Dror G. Feitelson, Dan Tsafrir, and David Krakov, "Experience with using the parallel workloads archive," *Journal of Parallel and Distributed Computing*, vol. 74, no. 10, pp. 2967 – 2982, 2014.

[16] Meikel Poess, Raghunath Othayoth Nambiar, Kushagra Vaid, John M Stephens Jr, Karl Huppler, and Evan Haines, "Energy benchmarks: a detailed analysis," in *Proceedings of the 1st International Conference on Energy-Efficient Computing and Networking*. ACM, 2010, pp. 131–140.

[17] David A Lifka, "The anl/ibm sp scheduling system," in *Job Scheduling Strategies for Parallel Processing*. Springer, 1995, pp. 295–303.

[18] Ahuva W. Mu'alem and Dror G. Feitelson, "Utilization, predictability, workloads, and user runtime estimates in scheduling the ibm sp2 with backfilling," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 12, no. 6, pp. 529–543, 2001.

[19] Vineet Khare, Xin Yao, and Kalyanmoy Deb, "Performance scaling of multi-objective evolutionary algorithms," in *Evolutionary Multi-Criterion Optimization*. Springer, 2003, pp. 376–390.

[20] Juan J. Durillo, Vlad Nae, and Radu Prodan, "Multi-objective energy-efficient workflow scheduling using list-based heuristics," *Future Generation Computer Systems*, vol. 36, pp. 221 – 236, 2014.

[21] "Specpower_ssj2008 results," `http://www.spec.org/power_ssj2008/results/`, Acceso: 2019-05-11.
[22] "Spec power and performance benchmark methodology v2.1," `https://www.spec.org/power/docs/SPEC-Power_and_Performance_Methodology.pdf`, Acceso: 2019-05-11.
[23] Susanne Albers and Antonios Antoniadis, "Race to idle: New algorithms for speed scaling with a sleep state," *ACM Trans. Algorithms*, vol. 10, no. 2, pp. 9:1–9:31, Feb. 2014.
[24] S. Pakin and M. Lang, "Energy modeling of supercomputers and large-scale scientific applications," in *Green Computing Conference (IGCC), 2013 International*, June 2013, pp. 1–6.
[25] Achim Streit, *Self-tuning job scheduling strategies for the resource management of HPC systems and computational grids*, Ph.D. thesis, Padeborn University, Germany, 2003.
[26] Henan Zhao and Rizos Sakellariou, "An experimental investigation into the rank function of the heterogeneous earliest finish time scheduling algorithm," in *Euro-Par 2003 Parallel Processing*, Harald Kosch, László Böszörményi, and Hermann Hellwagner, Eds., vol. 2790 of *Lecture Notes in Computer Science*, pp. 189–194. Springer Berlin Heidelberg, 2003.
[27] Gade Pandu Rangaiah, *Multi-objective optimization: techniques and applications in chemical engineering*, vol. 1, World Scientific, 2008.
[28] Abdullah Konak, David W Coit, and Alice E Smith, "Multi-objective optimization using genetic algorithms: A tutorial," *Reliability Engineering & System Safety*, vol. 91, no. 9, pp. 992–1007, 2006.
[29] Carlos A Coello Coello, "Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: a survey of the state of the art," *Computer methods in applied mechanics and engineering*, vol. 191, no. 11, pp. 1245–1287, 2002.
[30] Srividya Srinivasan, Rajkumar Kettimuthu, Vijay Subramani, and P Sadayappan, "Characterization of backfilling strategies for parallel job scheduling," in *Parallel Processing Workshops, 2002. Proceedings. International Conference on*. IEEE, 2002, pp. 514–519.

Algoritmo descentralizado para la asignación de servicios en arquitecturas de *Fog Computing* basado en un proceso expansivo de migración de instancias

Isaac Lera, Carlos Guerrero y Carlos Juiz[1]

Resumen— **Las tecnologías *Fog Computing* se han definido para mejorar el tiempo de respuesta y reducir el uso de la red de las aplicaciones del Internet de las Cosas que se basan en el uso de servicios de la nube. Para ello, permiten la ejecución de servicios y aplicaciones en nodos de comunicación más cercanos a los usuarios. Esto conlleva la necesidad de establecer políticas de asignación de servicios a los nodos de computación *Fog*. Dado que los entornos del Internet de las Cosas están compuestos por un gran número de componentes, las soluciones basadas en el conocimiento global del sistema tienen problemas de escalabilidad. Por eso, proponemos usar una política que, basándose únicamente en el conocimiento local de cada nodo *Fog* y de cada servicio, sea capaz de trazar una estrategia para escalar y aproximar los servicios a todos los usuarios que los solicitan. Este proceso de expansión considera que cada servicio ha de ser capaz de escalarse de forma autónoma y migrar hacia los siguientes nodos más cercanos en cada una de las rutas de red por las que se reciben peticiones de los usuarios. Esta política ha sido implementada en un simulador y se ha comprobado como la asignación de los servicios llega a un estado estacionario en el que los tiempos de respuesta van disminuyendo.**

Palabras clave— **Fog computing, Asignación de servicios, Optimización, Internet de las Cosas.**

I. Introducción

En los últimos años, se ha producido un avance significativo en las tecnologías relacionadas con el Internet de las Cosas (*Intenert of Thing*, IoT) y en su combinación con las tecnologías de Computación en la Nube (*Cloud Computing*) [1], [2]. La combinación de ambas tecnologías permite, en primer lugar, que las aplicaciones IoT dispongan de capacidades computacionales y de almacenamiento ilimitadas y, en segundo lugar, ampliar el alcance de las aplicaciones basadas en la nube al interactuar con componentes de la vida real [3], [4]. Pero la integración de IoT y la nube genera nuevos problemas, como por ejemplo el aumento en la latencia del servicio que es un requisito crítico para algunas aplicaciones IoT, como por ejemplo en el ámbito del *e-Health* o los juegos [5]. La tecnología conocida con el nombre de *Fog Computing* surgió para cubrir estas limitaciones y abrió una amplia gama de nuevos retos en temas como seguridad, confiabilidad, sostenibilidad, escalado, o gestión de recursos [6], [7], [8].

Las tecnologías *Fog Computing* se basan en disponer de capacidades de almacenamiento y ejecución en los recursos de los componentes de red y, de esta forma, poder asignar servicios o almacenar datos en las capas intermedias de la comunicación entre los usuarios IoT y el Cloud. Así se consigue que, en primer lugar, los servicios estén más cerca de los clientes y, en segundo lugar, los datos no necesitan ser transferidos en su totalidad a la nube. Ambos problemas tienen un impacto notable en el rendimiento, reduciendo la latencia y el uso de la red. Por tanto, la asignación de recursos a servicios toma un papel clave en la mejora del rendimiento.

Los algoritmos de asignación de recursos en arquitecturas *Fog* tienen la responsabilidad de seleccionar los nodos *Fog* (nodos de red intermedios con capacidades de computación) que se encargarán de ejecutar los servicios que solicitan los usuarios del sistema. Estos algoritmos deberán implementar las políticas que consigan encontrar la asignación y el nivel de escalado de los servicios que mejore los objetivos perseguidos (calidad de servicio, disponibilidad, rendimiento, etc.) [9].

Los entornos IoT, como por ejemplo las ciudades inteligentes, suelen tener miles, incluso millones, de dispositivos, usuarios y servicios. En estos niveles a gran escala, una gestión centralizada podría llegar a ser inabordable o de baja calidad. Pero un rápido repaso de la bibliografía reciente, nos muestra que estas soluciones centralizadas son las más habituales para solucionar el problema de la asignación de servicios *Fog* (*Fog Service Placement Problem*, FSPP). Los inconvenientes principales de una solución centralizada son: (a) escalabilidad, ya que el tiempo de ejecución de un algoritmo de optimización global generalmente aumenta a medida que aumenta la cantidad de dispositivos a administrar; (b) sobrecarga de la red, ya que los dispositivos deben enviar sus datos de rendimiento al gestor centralizado; (c) confiabilidad, ya que el gestor de recursos es un punto único de fallo (*Single Point Of Failure*, SPOF); y (d) latencia, ya que las decisiones deben transmitirse desde el gestor centralizado a cada dispositivo.

En este trabajo, proponemos solucionar el problema de la asignación de servicios a nodos *Fog* mediante una política descentralizado, de forma que cada servicio, mediante el uso de reglas generales e idénticas para todo el sistema, tomen decisiones sobre el nivel de escalado y la asignación y migración del servicio, en base a la información local de cada nodo.

[1]Dpto. de Matemàtiques i Informàtica, Univ. Illes Balears, e-mail: {isaac.lera|carlos.guerrero|cjuiz}@uib.es.

De esta forma, la solución es fácilmente aplicable a sistemas con un gran número de dispositivos, ya que los datos de rendimiento no se envían entre dispositivos, se evitan las SPOF y las decisiones se toman localmente. Proponemos basar las decisiones del algoritmo con el objetivo de ubicar los servicios más populares lo más cerca posible de los clientes.

El artículo está organizado de la siguiente forma: en la Sección II se presenta un breve repaso de trabajos relacionados con la asignación de servicios a nodos de computación en sistemas de *Fog Computing*; la Sección III está dedicada a definir formalmente el problema de la asignación de servicios a nodos *Fog*; en la Sección IV definimos los detalles de la política de asignación que proponemos en este estudio; y evaluamos nuestra política en un caso genérico mediante simulación en la Sección V. Finalmente, en la Sección VI presentamos las conclusiones y los posibles trabajos futuros que surgen de este primer trabajo exploratorio.

II. Trabajos relacionados

A pesar de la novedad de las tecnologías *Fog*, ya existe un gran número de trabajos que abordan el problema de la asignación de los servicios a nodos *Fog* [10]. Dado que este es un problema NP-completo, la mayoría de las soluciones de optimización se han abordado con el uso de heurísticas tales como algoritmos voraces, programación lineal o algoritmos genéticos, entre otros. Todos estos trabajos llevan a cabo la optimización del sistema analizándolo de una forma global. Y hay muy pocos estudios que se basen en una gestión local de los recursos, basada en algoritmos descentralizados.

La programación lineal es un enfoque común para la optimización de recursos. Arkian et al. [11] formularon un programa no lineal de enteros mixtos que se transformó en un programa lineal para la optimización del coste. Gu et al. [12] también utilizó este enfoque de optimización. Integraron los sistemas médicos cibernéticos en un sistema de *Fog Computing* y optimizaron el costo considerando la asociación de la estación base, la distribución de tareas y la ubicación de las máquinas virtuales. Finalmente, el trabajo de Velásquez et al. [13] usó programación lineal para reducir la cantidad de migraciones de servicios y la latencia de la red.

Huang et al. [14] presentó una formulación de programación cuadrática para el problema de reducir el consumo de energía en arquitecturas *Fog* mediante la asignación conjunta de servicios vecinos en los mismos dispositivos. Huang et al. [15] presentó un trabajo similar al anterior en el que el problema se modeló como un problema de conjunto de valores independientes máximo (MWIS). Souza et al. [16] estudiaron un algoritmo de asignación basado en la Programación Lineal Integral que minimizaba las latencias del servicio, garantizando el cumplimiento de los requisitos de capacidad. Skarlat et al. [17] estudió el problema de ubicación del servicio considerando los requisitos de QoS de las aplicaciones ejecutadas. Zeng et al. [18] propuso administrar la programación de tareas, la ubicación del almacenamiento de tareas y el uso equilibrado de Entrada/Salida para reducir el tiempo de ejecución de tareas. El trabajo de Barceló et al. [19] se centró en reducir el consumo de energía en entornos de IoT solucionado a través de un problema de flujo de costo mínimo mixto.

Un segundo conjunto presentamos los trabajos basados en algoritmos genéticos (*Genetic Algorithms*,GA). Wen et al. [9] presentaron un GA paralelo para reducir el tiempo de respuesta. Skarlat et al. [20] introdujo el concepto de colonias de dispositivos *Fog* para un proceso de optimización jerárquico. Cada colonia usó un GA para decidir los servicios que se asignaba a la colonia y cuáles se propagaron a las colonias vecinas. Yang et al. [21] compararon tres algoritmos de optimización basados en un algoritmo voraz, programación lineal y un GA. Además, también se presentó un modelo para predecir la distribución de las solicitudes futuras del usuario para adaptar la asignación del servicio. Los autores de este propuesta también han abordado el problema de la asignación de servicios a dispositivos *Fog* mediante el uso de algoritmos genéticos [22]. En ese trabajo comparamos tres algoritmos genéticos y estudiamos las ventajas que ofrecía cada uno de ellos en frente de los demás.

Como ya hemos indicado, todos estos trabajos abordan la solución desde un punto de vista global del sistema. Pero esto genera limitaciones como la escalabilidad del problema (derivando en tiempo de ejecución muy altos para encontrar soluciones a la asignación de servicios), como el aumento del tráfico de red ya que se ha de enviar al gestor centralizado toda la información del sistema. Por eso, presentamos una primera solución descentralizada al problema de la asignación de servicios *Fog* [23].

III. Definición del problema

Las tecnologías de *Fog computing* son un patrón de arquitectura en el que los clientes solicitan servicios a proveedores de la nube a través de una red compuesta por dispositivos de red. Estos dispositivos, llamados dispositivos *Fog*, tienen capacidades computacionales y de almacenamiento que les permiten almacenar datos y ejecutar servicios que tradicionalmente se han llevado a cabo en la nube. Esto genera la necesidad de definir políticas de administración de datos y servicios para decidir cuándo y dónde colocar los servicios y los datos. Nuestra propuesta de arquitectura se centra en el problema de asignación de servicios a para ser ejecutados en dispositivos *Fog* (*Fog Service Placement Problem*, FSPP).

En la Figura 1 se representa la arquitectura general de un sistema de *Fog Computing*, donde se pueden identificar las tres capas comunes: *Cloud*, *Fog* y clientes. La arquitectura se puede modelar como un grafo donde los nodos son los dispositivos y las aristas las conexiones de red directas entre dispositivos. Se pueden diferenciar tres tipos de dispositivos: un dispositivo para el proveedor *Cloud*; las pasarelas

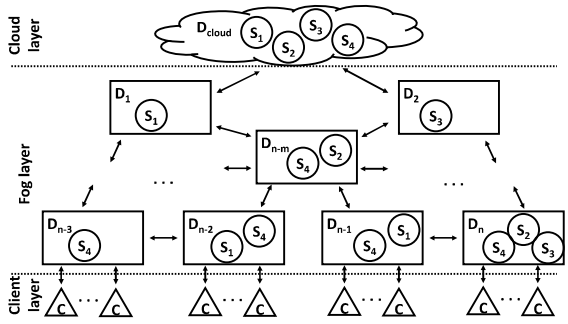


Fig. 1. Arquitectura de un sistema de *Fog Computing*.

(o *Gateways*), que son los puntos de acceso para los clientes; y los dispositivos *Fog*, los dispositivos de red entre el *Cloud* y los *Gateways*. Todos estos dispositivos tienen recursos que pueden ser asignados para la ejecución de servicios.

De una manera más formal, el FSPP considera un conjunto de clientes, C_n, que solicitan aplicaciones que están alojadas en un proveedor *Cloud* ,D_{cloud}. Las aplicaciones se modelan como un único servicio, S_x. Los clientes solicitan estas aplicaciones a través de un conjunto de dispositivos de red interconectados, D_i. Estos dispositivos tienen recursos que les permiten la ejecución de los servicios *Cloud*. Las conexiones físicas de los nodos o dispositivos se pueden definir como una estructura de grafo donde los dispositivos son los nodos y las conexiones entre dispositivos son las aristas.

Un mismo servicio puede estar instanciado en distintos dispositivos, S_x^y, permitiendo así la escalabilidad de las aplicaciones que estarán por tanto definidas a través de un patrón de aplicaciones *Stateless* [24]. La función de asignación de un servicio o aplicación a un dispositivo *Fog* es, por tanto, una relación muchos-a-muchos, $alloc : \{S_x\} \rightarrow \{D_i\}$.

Los clientes están conectados al sistema a través de dispositivos que incorporan dicha capacidad, conocidos habitualmente como *Gateways*. Estos clientes pueden ser dispositivos móviles, sensores, actuadores, entre otros. Estas conexiones podemos modelarlas a través de una relación muchos-a-uno, $conn : \{C_n\} \rightarrow \{D_i\}$. Cada uno de estos clientes está caracterizado por la frecuencia de peticiones, $\lambda_{S_x}^{C_n}$. Por tanto, cada dispositivo del sistema puede caracterizarse por la frecuencia de peticiones que recibe para cada uno de los servicios que tiene alojado, $\lambda_{S_x}^{D_i}$, y es capaz igualmente de discernir el nodo por el que le llegan dichas peticiones, $\lambda_{S_x}^{D_i \leftarrow D_{i'}}$.

Adicionalmente, estos dispositivos también se pueden caracterizar por su capacidad, es decir, los recursos de computación que ofrecen. De forma general, se pueden definir como un conjunto de n valores, donde cada uno corresponden a la capacidad de cada uno de sus recursos (procesador, memoria, almacenamiento, etc.), $R_{D_i}^{cap} =< r_0, r_1, ...r_{n-1} >$.

Los servicios asignados a un dispositivo generan un consumo de recursos, que podemos definir también como una tupla, $R_{S_x}^{con} =< r_0, r_1, ...r_{n-1} >$. El consumo total de recursos en un dispositivo se calcula como la suma de los recursos consumidos por cada uno de los servicios asignados a dicho recurso:

$$R_{D_i}^{u} = \sum^{S_x} R_{S_x}^{con} \quad \forall\ S_x \mid S_x \ \in \ alloc(D_i) \qquad (1)$$

Por tanto, existe una restricción en la asignación de los servicios a los dispositivos y es que la suma de recursos consumidos por los servicios asignados debe de ser menor a los recursos disponibles en el dispositivo.

IV. Propuesta de algoritmo

En este trabajo proponemos definir una política de asignación de recursos a servicios *Fog* inspirada en un proceso de concurrencia competitiva juntamente con una evolución expansiva y colonizadora que aparece en muchos fenómenos naturales. Por ejemplo, el proceso por el que las especies animales se adaptan a distintos ecosistemas. Las especies biológicas que se encuentran en un mismo ecosistema compiten por los nichos ecológicos presentes, hasta que se estabilizan las distintas poblaciones. Para ello, las distintas especies compiten por unos recursos limitados (terreno, alimentos, etc.), intentando todas ellas ocupar el mayor número de nichos biológicos posibles y que mejor cubran sus necesidades. Este proceso llegará rápidamente a un proceso de equilibrio en el que las especies supervivientes estarán casi completamente adaptadas al nuevo ambiente. El proceso de adaptación volverá a iniciarse cada vez que se produzca un cambio ambiental en un ecosistema o la introducción de una nueva especie en el mismo.

En nuestra propuesta, hemos establecido una analogía entre este proceso biológico y el problema que abordamos en nuestro estudio, podríamos decir que los servicios corresponden a las especies biológicas, los dispositivos *Fog* son los nichos ecológicos, y el nivel de escalado de los servicios y su asignación a los nodos corresponde a los nichos que ocuparán las distintas especies en un ecosistema.

Igualmente, hemos fijado que los objetivos de optimización de nuestra política de asignación de recursos a servicios son minimizar la latencia de los servicios y aumentar la disponibilidad. Bajo estos requisitos, una estrategia adecuada es asignar un servicio en cada uno de los nodos en los que se encuentren conectados usuarios que solicitan dicho servicio. Dado que esto solo sería posible en un sistema irrealista donde los recursos de los nodos son ilimitados, o bien, solo existe un tipo de servicio, se producirá una concurrencia competitiva entre servicios que intentarán ser asignados en un mismo nodo con recursos limitados.

Los servicios, o en su defecto los nodos que los tengan asignados, deberán decidir de forma autónoma e independiente llevar a cabo una serie de operaciones (escalar, migrar o no llevar a cabo ninguna operación) en función de las condiciones locales de dicho nodo. En nuestra propuesta, hemos considerado que la información utilizada para evaluar las condiciones

locales de un nodo son el número de peticiones de un servicio que llegan a dicho nodo, y el nodo directamente conectado desde el que llegan dichas peticiones (variables independientes). Considerando estos conjuntos de variables independientes y de operaciones, definimos las siguientes reglas para cada uno de los casos identificados:

- Regla de colonización. Dado un nodo, este analiza cada uno de los servicios que tiene asignados, e intenta generar nuevas instancias del servicio en todos los nodos desde los que se reciben peticiones a dicho servicio. Dicho de otra manera, se intentan instanciar servicios en todos los caminos por los que llegan actualmente las peticiones al nodo. En la Fig. 2 (a) vemos como el servicio de color verde se propagará a los nodos por donde se le soliciten peticiones.
- Regla de colisión, cuando los servicios que se están intentado asignar a un nodo, consumen más recursos de los que tiene el propio nodo, se debe de aplicar una regla de selección para decidir que servicios son asignados y cuales rechazados. Para ello se establece una regla que selecciona aquellos servicios que tengan una tasa de peticiones más elevada. Los servicios que sean rechazados o desalojados migrarán al nodo inmediatamente anterior al que estuvieron asignados. En la Fig. 2 (b), el servicio de color verde tiene una mayor preferencia sobre el servicio de color azul debido al mayor número de peticiones recibidas por ese nodo. Por lo tanto, se desplegará sobre ese nodo y desplazará al servicio azul.
- Regla de inanición. Si un servicio resulta estar asignado en un nodo en el que se detecta que no se reciben más peticiones de dicho servicio, este servicio es eliminado. En la Fig. 2 (c), el servicio ubicado en la cúspide de esta disposición no recibe peticiones de ningún enlace, por lo tanto será eliminado.

Es importante aclarar que, aunque se ha comentado que los servicios tendrán un comportamiento autónomo e independiente, este comportamiento no puede estar programado dentro del propio servicio. Obviamente, es el propio dispositivo *Fog* el que presenta un rol activo en la orquestación de los servicios, y es el elemento que implementa las reglas. Por eso, las reglas han sido definidas desde el punto de vista del nodo.

V. Experimentación

Para ejecutar los experimentos para validar nuestra política de asignación, hemos decidido utilizar un entorno de simulación. Para ello, hemos seleccionado el simulador YAFS (*Yet Another Fog Simulator*) que hemos desarrollado previamente para otros estudios de investigación. Dicho simulador, permite definir topologías de red basadas en grafos y redes complejas, además de que permite la definición de escenarios dinámicos que evolucionan a lo largo de la simulación, e implementar políticas de asignación de servicios a nodos que analicen estos cambios durante dicha simulación. El simulador está disponible como código libre y dispone de un gran número de ejemplos de uso [25].

Para esta experimentación, hemos seleccionado una topología de red basada en una cuadrícula de 10x10 (100 nodos) donde los nodos diagonales también han sido conectados. Dado que hemos considerado un ejemplo genérico, todos los parámetros están definidos en unidades genéricas de tiempo (t). De esta forma, hemos fijado que la latencia de los enlaces entre nodos es igual en todo el sistema y tiene un valor de 1 t. Para maximizar el número de colisiones durante la simulación, también hemos supuesto que solo es posible asignar un servicio a cada uno de los nodos.

Hemos supuesto que los servicios se encuentran disponibles inicialmente en dos proveedores de *Cloud* que se encuentran conectados en los dos nodos de los vértices superiores de la cuadrícula. Por el contrario, los usuarios se sitúan cubriendo todos los nodos del lado inferior de la cuadrícula. Hemos supuesto que un usuario solo solicita uno de los servicios y que todos los usuarios generan la misma carga sobre el sistema (el mismo número de peticiones por unidad de tiempo). Hemos fijado el tiempo entre peticiones de un usuario en 500 t. Para poder ver más claramente como la asignación de los servicios evoluciona hacia las zonas en las que estos son solicitados con una mayor frecuencia, hemos supuesto que el número de usuarios de cada tipo de servicio crece linealmente a lo largo de los nodos que ocupan el lado inferior de la cuadrícula. De esta forma, el nodo del vértice inferior derecho tiene 9 usuarios que solicitan el primer servicio (verde), y estos van decreciendo de uno en uno hasta el nodo inferior izquierdo donde no tiene ningún usuario conectado. El segundo servicio (naranja) tiene la misma distribución, pero en sentido contrario. Si nos fijamos en las figuras de la Figura 3, podemos ver que los proveedores *Cloud* se encuentran conectados en los nodos 90 y 99, y que los usuarios se encuentran conectados en los nodos que van desde el 0 hasta el 9. Los números de la parte inferior indican el número de usuarios para cada uno de los servicios en cada uno de esos 10 nodos. Con está disposición, donde los proveedores del *Cloud* y los solicitantes del servicio se encuentran en lados opuestos, se busca de nuevo maximizar las colisiones entre las asignaciones de los servicios.

La simulación de nuestro experimento, como ya hemos dicho anteriormente, se inicia asignando los dos servicios disponibles en el nodo 90 y el nodo 99 de la red, y no realiza ningún cambio sobre esta asignación durante los primeros 5×10^5 t. A partir de ese momento, se evalúan modificaciones en la asignación de los servicios en intervalos de $2,5 \times 10^4$ t, y se realiza ya de forma ininterrumpida hasta el final de la simulación. La Figura 3 muestra la asignación de los servicios para cada uno de estos instantes de tiempo o intervalos donde hay variación en la ubicación de los servicios. Cada destacar que en esta experimen-

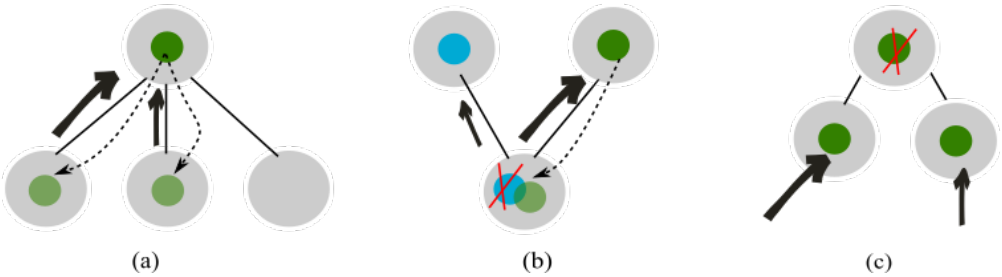


Fig. 2. Reglas de colonización (a), de colisión (b) y de inanición (c).

tación, las activaciones son periódicas, pero pueden ser autónomas e independientes en cada servicio.

Del análisis de la Figura 3 podemos observar, en primer lugar, que a medida que se van realizando evaluaciones de la asignación de los servicios, estos se van aproximando cada vez más a la parte inferior de la red y como se produce una intersección de sus caminos para llegar a los nodos donde tienen un mayor número de usuarios. También observamos que los servicios se han escalado, de forma que el algoritmo no solo es capaz de gestionar la asignación de los servicios, sino que también es capaz de aumentar el nivel de escalado de los servicios de forma autónoma.

Otra observación de la evolución del experimento es su falta de simetría. Este hecho se debe a la implementación de la propagación del número de peticiones entre los servicios y los usuarios. De manera aleatoria se elige un camino mínimo del conjunto de existentes para conectar servicios y usuarios, lo cual conllevará a agrupar peticiones y, por ende, activar o inhibir servicios. Así, en la Figura 3, paso 2 -esquina superior derecha-, existe un servicio verde que está activo a diferencia de su homólogo naranja. Este criterio de enrutamiento elegido aleatoriamente persistirá hasta que no haya un mejor servicio al cual acceder en términos de distancia.

Finalmente, y aunque no está reflejado en las gráficas, se observó que después del intervalo 10, a pesar de que se siguieron realizando evaluaciones de la asignación de los servicios, la asignación de estos ya no variaba. Por lo tanto, se ve como nuestra política converge a un estado estacionario en el que no se volverán a realizar cambios en la asignación de servicios a nodos. Esto será así mientras que no se produzca ningún cambio en el sistema (número de usuario, fallos en los nodos, cambio de la topología, etc.) o no se desplieguen nuevos servicios en el sistema. Sería interesante, y queda como trabajo futuro, poder evaluar el impacto de este tipo de modificaciones en el sistema.

Por último, también hemos analizado la evolución del tiempo de respuesta de los servicios a medida que estos van cambiando de ubicación. De esta forma, la Figura 4 muestra el tiempo de respuesta medio de todas las peticiones que reciben los servicios a medida que van avanzando los intervalos de evaluación de la asignación de recursos (o pasos). Vemos que, tal y como era de esperar, el tiempo de respuesta disminuye a medida que los servicios van siendo asignados a nodos más cercanos a los usuarios.

VI. Conclusiones

En este estudio, se ha explorado la viabilidad de una propuesta descentralizada de ubicación de servicios en escenarios de *Fog Computing*. Esta política está inspirada en procesos de concurrencia competitiva presente en seres que compiten por recursos limitados en ecosistemas compartidos. Una propuesta descentralizada disminuye el impacto del análisis de las posibles alternativas y cambios de todas las variables en modelos de optimización que consideran globalmente el sistema. En nuestro caso, los servicios, a través de los nodos, muestran un comportamiento que se basa en tres reglas: colonización, colisión e inanición. En el experimento presentado se analiza como el entorno bajo una región de estabilidad converge en una ubicación donde se reduce los tiempos de latencia y el tráfico en la red. Es decir, los resultados están en línea con los objetivos perseguidos con la implantación de arquitecturas *Fog Computing*. Como trabajo futuro, consideramos importante explorar otras topologías de red. Igualmente es necesario llevar a cabo experimentación con escenarios dinámicos en los que tanto el sistema sea modificado (movimiento de usuarios y su comportamiento, fallo de nodos, etc.) como la cantidad de servicios en el sistema vayan cambiando a lo largo de la experimentación.

Agradecimientos

Este trabajo ha sido financiado por el Gobierno de España (Agencia Estatal de Investigación) y la Comisión Europea (Fondo Europeo de Desarrollo Regional) a través del proyecto TIN2017-88547-P (MINECO/AEI/FEDER, UE).

Referencias

[1] Alessio Botta, Walter de Donato, Valerio Persico, and Antonio Pescape, "Integration of cloud computing and internet of things: A survey," *Future Generation Computer Systems*, vol. 56, no. Supplement C, pp. 684 – 700, 2016.

[2] Manuel Diaz, Cristian Martin, and Bartolome Rubio, "State-of-the-art, challenges, and open issues in the integration of internet of things and cloud computing," *Journal of Network and Computer Applications*, vol. 67, no. Supplement C, pp. 99 – 117, 2016.

[3] Everton Cavalcante, Jorge Pereira, Marcelo Pitanga Alves, Pedro Maia, Roniceli Moura, Thais Batista, Flavia C. Delicato, and Paulo F. Pires, "On the interplay of internet of things and cloud computing: A systematic mapping study," *Computer Communications*, vol. 89-90, no. Supplement C, pp. 17–33, 2016, Internet of Things: Research challenges and Solutions.

[4] Ashraf Darwish and Aboul Ella Hassanien, "Cyber physical systems design, methodology, and integration: the

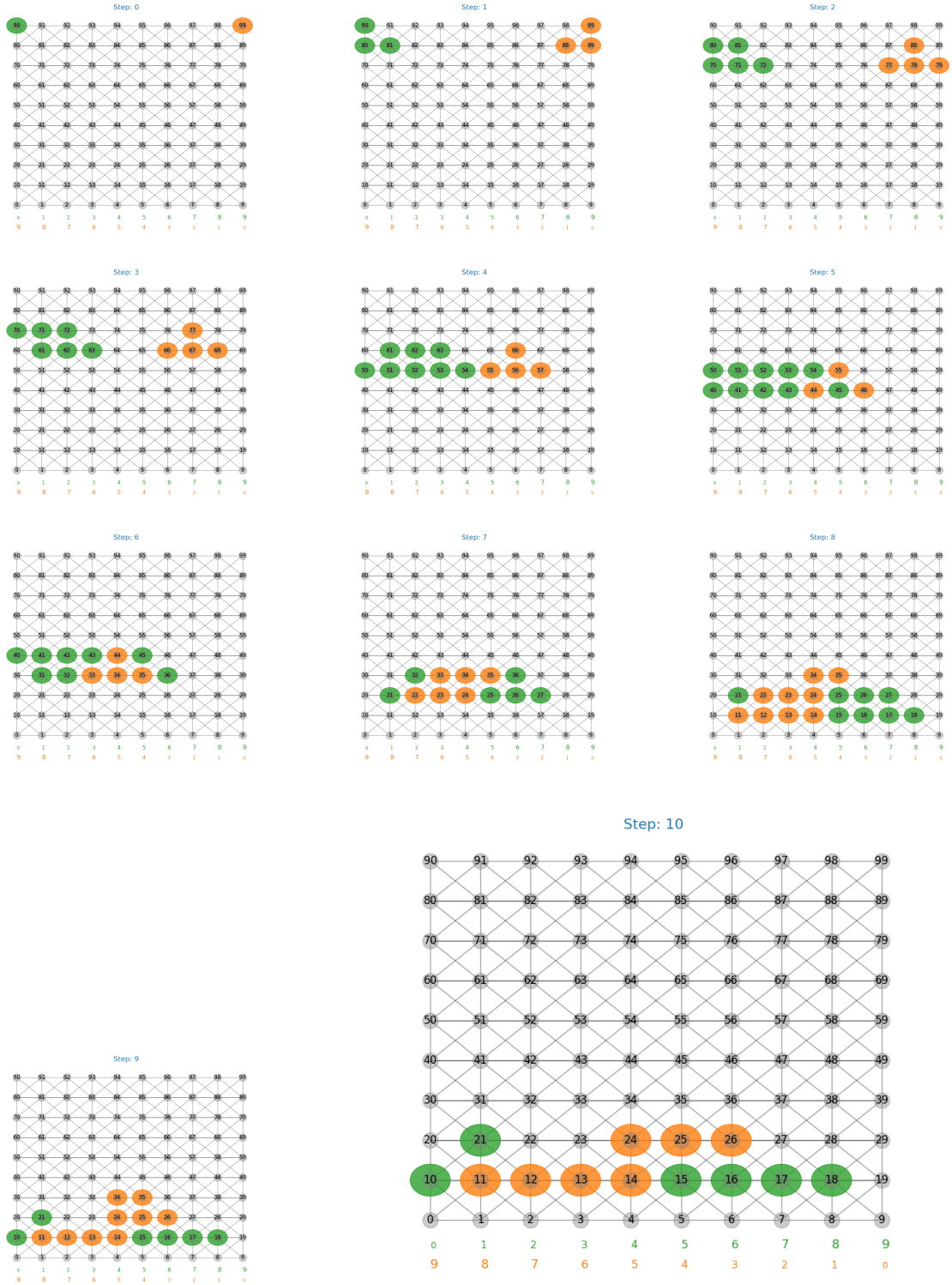


Fig. 3.Evolución de la asignación de los servicios. La posición inicial corresponde a la esquina superior y su evolución temporal es de izquierda a derecha y de arriba a abajo.

current status and future outlook," *Journal of Ambient Intelligence and Humanized Computing*, Sep 2017.

[5] A. V. Dastjerdi and R. Buyya, "Fog computing: Helping the internet of things realize its potential," *Computer*, vol. 49, no. 8, pp. 112–116, Aug 2016.

[6] Blesson Varghese and Rajkumar Buyya, "Next generation cloud computing: New trends and research directions," *Future Generation Computer Systems*, 2017.

[7] Redowan Mahmud, Ramamohanarao Kotagiri, and Rajkumar Buyya, *Fog Computing: A Taxonomy, Survey and Future Directions*, pp. 103–130, Springer Singapore, Singapore, 2018.

[8] M. Chiang and T. Zhang, "Fog and iot: An overview of research opportunities," *IEEE Internet of Things Journal*, vol. 3, no. 6, pp. 854–864, Dec 2016.

[9] Z. Wen, R. Yang, P. Garraghan, T. Lin, J. Xu, and M. Rovatsos, "Fog orchestration for internet of things services," *IEEE Internet Computing*, vol. 21, no. 2, pp. 16–24, Mar 2017.

[10] Antonio Brogi, Stefano Forti, Carlos Guerrero, and Isaac Lera, "How to place your apps in the fog - state of the art and open challenges," *CoRR*, vol. abs/1901.05717, 2019.

[11] Hamid Reza Arkian, Abolfazl Diyanat, and Atefe Pourkhalili, "Mist: Fog-based data analytics scheme with cost-efficient resource provisioning for iot crowdsensing appli-

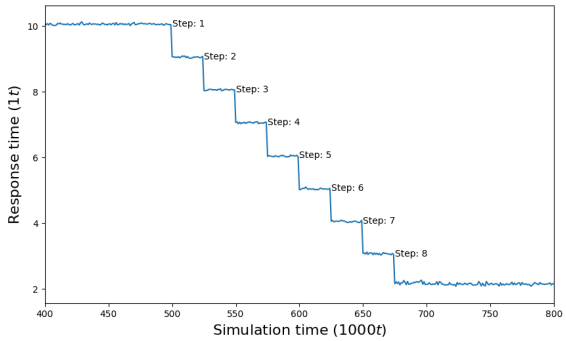


Fig. 4.Evolución del tiempo de respuesta resaltando las activaciones del conjunto de reglas

cations," *Journal of Network and Computer Applications*, vol. 82, no. Supplement C, pp. 152 – 165, 2017.

[12] L. Gu, D. Zeng, S. Guo, A. Barnawi, and Y. Xiang, "Cost efficient resource management in fog computing supported medical cyber-physical system," *IEEE Transactions on Emerging Topics in Computing*, vol. 5, no. 1, pp. 108–119, Jan 2017.

[13] Karima Velasquez, David Perez Abreu, Marilia Curado, and Edmundo Monteiro, "Service placement for latency reduction in the internet of things," *Annals of Telecommunications*, vol. 72, no. 1, pp. 105–115, Feb 2017.

[14] Zhenqiu Huang, Kwei-Jay Lin, Shih-Yuan Yu, and Jane Yung jen Hsu, "Co-locating services in iot systems to minimize the communication energy cost," *Journal of Innovation in Digital Ecosystems*, vol. 1, no. 1, pp. 47 – 57, 2014.

[15] Zhenqiu Huang, Kwei-Jay Lin, Shih-Yuan Yu, and Jane Yung-jen Hsu, "Building energy efficient internet of things by co-locating services to minimize communication," in *Proceedings of the 6th International Conference on Management of Emergent Digital EcoSystems*, New York, NY, USA, 2014, MEDES '14, pp. 18:101–18:108, ACM.

[16] V. B. C. Souza, W. Ramírez, X. Masip-Bruin, E. Marín-Tordera, G. Ren, and G. Tashakor, "Handling service allocation in combined fog-cloud scenarios," in *2016 IEEE International Conference on Communications (ICC)*, May 2016, pp. 1–5.

[17] O. Skarlat, S. Schulte, M. Borkowski, and P. Leitner, "Resource provisioning for iot services in the fog," in *2016 IEEE 9th International Conference on Service-Oriented Computing and Applications (SOCA)*, Nov 2016, pp. 32–39.

[18] D. Zeng, L. Gu, S. Guo, Z. Cheng, and S. Yu, "Joint optimization of task scheduling and image placement in fog computing supported software-defined embedded system," *IEEE Transactions on Computers*, vol. 65, no. 12, pp. 3702–3712, Dec 2016.

[19] M. Barcelo, A. Correa, J. Llorca, A. M. Tulino, J. L. Vicario, and A. Morell, "Iot-cloud service optimization in next generation smart environments," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 12, pp. 4077–4090, Dec 2016.

[20] Olena Skarlat, Matteo Nardelli, Stefan Schulte, Michael Borkowski, and Philipp Leitner, "Optimized iot service placement in the fog," *Service Oriented Computing and Applications*, Oct 2017.

[21] L. Yang, J. Cao, G. Liang, and X. Han, "Cost aware service placement and load dispatching in mobile cloud systems," *IEEE Transactions on Computers*, vol. 65, no. 5, pp. 1440–1452, May 2016.

[22] Carlos Guerrero, Isaac Lera, and Carlos Juiz, "Evaluation and efficiency comparison of evolutionary algorithms for service placement optimization in fog architectures," *Future Generation Computer Systems*, vol. 97, pp. 131 – 144, 2019.

[23] Carlos Guerrero, Isaac Lera, and Carlos Juiz, "A lightweight decentralized service placement policy for performance optimization in fog computing," *Journal of Ambient Intelligence and Humanized Computing*, vol. 10, no. 6, pp. 2435–2452, Jun 2019.

[24] B. I. Ismail, E. Mostajeran Goortani, M. B. Ab Karim, W. Ming Tat, S. Setapa, J. Y. Luke, and O. Hong Hoe, "Evaluation of docker as edge computing platform," in *2015 IEEE Conference on Open Systems (ICOS)*, Aug 2015, pp. 130–135.

[25] Isaac Lera and Carlos Guerrero, "YAFS, yet another fog simulator," https://github.com/acsicuib/YAFS, Accedido: Mayo 2019.

Un Simulador de Paralelismo de Modelo para Redes Neuronales

Adrián Castelló[1], Manuel F. Dolz[2], Enrique S. Quintana-Ortí[1], José Duato[1]

Resumen— **En este trabajo presentamos un simulador de rendimiento que comprende las principales etapas de cálculo y comunicaciones que aparecen en el entrenamiento supervisado de redes neuronales sobre un cluster paralelo. El simulador realiza un entrenamiento síncrono y explota el paralelismo de modelo por capas, de modo que la distribución del trabajo sobre la base de las entradas que recibe cada capa. Además, el simulador está parametrizado, de modo que puede configurarse para simular cualquier perceptron multicapa o red neuronal convolucional.**

Palabras clave— **Redes Neuronales, Entrenamiento Supervisado, Paralelismo de Modelo, Simulación, Clusters**

I. Introducción

En los últimos años, el aprendizaje automático (*machine learning*) ha transcendido de un nicho de problemas tradicionales en clasificación de imágenes y reconocomiento del habla [1–3], para convertirse en una tecnología clave en un abundante número de aplicaciones, tales como la computación cuántica, la iluminación de estados sólidos, la radiografía y la telegrafía, y la simulación astrofísica, por nombrar unos pocos casos [4–6].

La tecnología que alimenta los avances logrados mediante el aprendizaje automático es la red neuronal (RN), un algoritmo genérico que se adapta a sí mismo para resolver problemas específicos. En el aprendizaje supervisado, este proceso seguida a través de un entrenamiento previo que, en general, es computacionalmente costoso, especialmente para RNs con muchas capas/entradas/salidas (RNs profundas). Esta etapa inicial es sucedida por el uso de la RN entrada para resolver el problema (inferencia), que es relativamente económica, desde el punto de vista computacional, y a menudo puede realizarse sobre una circuitería de bajo coste [4, 7].

La mayor parte de los entornos para aprendizaje automático (por ejemplo, TensorFlow, Caffe2, etc.) explotan el *paralelismo de datos* para acelerar el proceso de aprendizaje sobre clusters de computadores. Sin embargo, aprovechar el *paralelismo de modelo* resulta fundamental cuando los modelos de RNs son tan grandes que no pueden replicarse en la memoria de los nodos del sistema. En estos casos resulta necesario aprovechar el paralelismo de modelo, que permite la distribución del modelo sobre los nodos del cluster. En este artículo introducimos nuestro simulador de paralelismo de modelo (SPM), que realiza las principales etapas computacionales y de cálculo que aparecen en perceptrones multicapa o red neuronal convolucionales, explotando el paralelismo de modelo sobre un cluster paralelo.

El resto del artículo está organizado del siguiente modo. En la Sección II ofrecemos una breve revisión del entrenamiento supervisado para RNs, y en la Sección III abordamos el paralelismo de modelo. A continuación, en la Sección IV presentamos el simulador SPM, y en la Sección V utilizamos SPM para evaluar el rendimiento de cuatro RNs representativas. Finalmente, en la Sección VI resumimos las contribuciones de este trabajo.

II. Entrenamiento de Redes Neuronales

A. Breve descripción

Consideremos un conjunto de entradas (o muestras), $x_1, x_2, \ldots \in \mathbb{R}^n$, clasificadas, para entrenamiento supervisado, con las etiquetas $y_1, y_2, \ldots \in \mathbb{R}^m$, respectivamente. Una RN define una función no lineal $\mathcal{F} : \mathbb{R}^n \to \mathbb{R}^m$ que establece una correspondencia $\mathcal{F}(x_r) = \tilde{y}_r$, donde podemos esperar que $\tilde{y}_r \approx y_r$, $r = 1, 2, \ldots$

La correspondencia entre entradas y salidas a través de una RN compuesta por L capas, con n_l neuronas en la capa $l = 1, 2, \ldots, L$, se define como:

$$\begin{aligned} a^{(l)} &= \sigma(z^{(l)}) = \\ &= \sigma(W^{(l)} a^{(l-1)} + b^{(l)}) \in \mathbb{R}^{n_l}, \\ &\quad l = 2, 3, \ldots, L, \end{aligned} \tag{1}$$

donde

- $a^{(1)} = x_r$ e $\tilde{y}_r = a^{(L)}$;
- $a^{(l)} = (a_j^{(l)}) \in \mathbb{R}^{n_l}$ son las salidas (activaciones) de las neuronas en la capa l;
- $W^{(l)} = (w_{ij}^{(l)}) \in \mathbb{R}^{n_l \times n_{l-1}}$ es la matriz de pesos en la capa l, con la entrada $w_{ij}^{(l)}$ asociada con la conexión entre la neurona j en la capa $l-1$ y la neurona i en la capa l;
- $b^{(l)} \in \mathbb{R}^{n_l}$ is el vector de desplazamientos (biases) en la capa l; y
- $\sigma(\cdot)$ es una función no lineal (p.e., sigmoide, ReLU, etc. [4]), que se aplica elemento a elemento sobre el vector de entrada.

La Figura 1 muestra una RN de cinco capas, con 4, 3, 5, 4 y 2 neuronas en las capas 1–5, respectivamente.

El proceso de entrenamiento persigue minimizar el error $\|y_r - \tilde{y}_r\|$ (para todo r), esto es, la diferencia entre las salidas calculadas por la RN como parte de la "propagación" (*forward pass*, FP) definida por (1) y las etiquetas asociadas. Este problema

[1]Departamento de Informática de Sistemas y Computadores, Universitat Politècnica de València, 46022–Valencia. E-mails: {adcastel,quintana,duato}@disca.upv.es

[2]Departamento de Ingeniería y Ciencia de los Computadores, Universitat Jaume I, 12.071–Castellón. E-mail: dolzm@uji.es

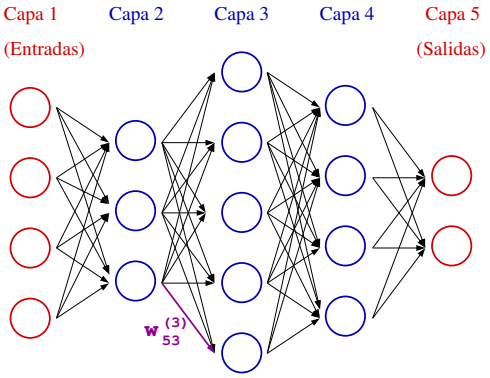


Fig. 1: Ejemplo de una red con 5 capas. Las capas 1 y 5 corresponden, respectivamente, a las entradas y salidas de la red. Las restantes capas están ocultas. La figura resalta la conexión entre la neurona 3 de la capa 2 y la neurona 5 de la capa 3, con peso $w_{53}^{(3)}$.

de optimización se resuelve habitualmente mediante del método SGD (*stochastic gradient descent*), que implementa una etapa de retroprogragación (*back-propagation*, BP) que calcula los gradientes de los pesos que minimizan el error. Estos valores se utilizan a continuación para actualizar los parámetros pesos y desplazamientos que definen el modelo en (1), en preparación para la siguiente iteración de SGD [8].

En la práctica, el entrenamiento se realiza por lotes (*batches*) de b entradas simultaneas, es decir $X = [x_r, x_{r+1}, \ldots, x_{r+b-1}] \in \mathbb{R}^{n_1 \times b}$. En caso de que b sea suficientemente grande, el uso de lotes permite eliminar el cuello de botella en el acceso a memoria para los núcleos computacionales que aparecen en el entrenamiento, haciendo que estas operaciones estén limitadas por la capacidad de procesamiento y no por la velocidad de acceso a memoria.

A continuación, revisamos el efecto de estas transformaciones en un perceptron multicapa, esto es, una red neuronal compuesta por capas completamente conectadas (*fully-connected*, FC). Las capas convolucionales (CONV) de una RN convolucional pueden tratarse del mismo modo, previa transformación de las activaciones de entrada a través, por ejemplo, de la función *im2col* [9] o de cualquiera de sus variantes propuestas recientemente [10, 11].

A.1 Perceptron multicapa

En el entrenamiento por lotes, los productos matriz-vector (1) que aparecen en las capas FP de este tipo de RNs se convierten en

$$Z^{(l)} = W^{(l)} A^{(l-1)} + B^{(l)}, \quad l = 2, 3, \ldots, L, \tag{2}$$

donde $Z^{(l)} \in \mathbb{R}^{n_l \times b}$ y $B^{(l)} = \left[b^{(l)}, b^{(l)}, \ldots, b^{(l)}\right] \in \mathbb{R}^{n_l \times b}$. Además, los cálculos de los gradientes durante la etapa BP-CG se transforman en un conjunto de operaciones de la forma:

$$G^{(l)} = (W^{(l+1)})^T G^{(l+1)}, \quad l = L-1, L-2, \ldots, 2, \tag{3}$$

TABLA I: Dimensiones de los operandos del producto GEMM $C = (C+)A \cdot B$.

	C	A	B
FP	$n_l \times b$	$n_l \times n_{l-1}$	$n_{l-1} \times b$
BP-GC	$n_l \times b$	$n_l \times n_{l+1}$	$n_{l+1} \times b$
BP-WU	$n_l \times n_{l-1}$	$n_l \times b$	$b \times n_{l-1}$

donde $G^{(l)} \in \mathbb{R}^{n_l \times b}$, $G^{(l+1)} \in \mathbb{R}^{n_{l+1} \times b}$, y $G^{(L)}$ se definen básicamente como una versión escalada de la diferencia entre la salida de la RN para el lote, es decir $A^{(L)} = \tilde{Y}$, y las respectivas etiquetas $Y = [y_r, y_{r+1}, \ldots, y_{r+b-1}]$. Finalmente, las actualizaciones de los pesos durante la propagación regresiva (BP-WU), se realizan mediante las operaciones:

$$W^{(l)} = W^{(l)} - \eta\, G^{(l)} (A^{(l-1)})^T, \quad l = L, L-1, \ldots, 2, \tag{4}$$

donde el escalar η representa el factor de aprendizaje [8].

En resumen, las tres multiplicaciones de matrices (GEMM) en (2)–(4) son de la forma $C = (C+)A \cdot B$, con las dimensiones de los operandos para la capa l mostrada en la Tabla I.

Las implementaciones optimizadas de las capas FC para procesadores multinúcleo (así como procesadores gráficos, GPUs) simplemente invocan a una rutina de altas prestaciones de la biblioteca BLAS, como las que pueden encontrarse en Intel MKL, OpenBLAS, GotoBLAS2, BLIS (o cuBLAS, para GPUs), para calcular el correspondiente producto matricial GEMM.

III. Paralelismo de Modelo sobre Clusters

En el entrenamiento síncrono, para la etapa FP la información fluye de izquierda a derecha en la RN, con estrictas dependencias entre capas adyacentes. Una vez esta etapa concluye, la etapa BP procede a calcular los gradientes y actualizar los pesos. Durante esta segunda etapa (compuesta por BP-GC y BP-WU), la información fluye entre capas adyacentes de derecha a izquierda, pero de nuevo con dependencias estrictas entre capas consecutivas. Esto impide una paralelización entre capas (a menos que se realice un entrenamiento asíncrono [12]) y, como consecuencia, la única opción es explotar el paralelismo interno en cada capa.

A. Paralelismo de modelo

En este trabajo, se asume una plataforma paralela compuesta por P procesos. Además, el simulador SPM explota el paralelismo de modelo, en el cual los parámetros que definen la RN (básicamente, los pesos) se particionan sobre estos procesos, de modo que inducen una distribución específica del trabajo [6].

En paralelismo de modelo, las tres multiplicaciones de matrices que aparecen en una determinada capa l, de tipo FC, se paralelizan particionando los datos de la capa (y distribuyendo la carga computacional) a lo largo de las entradas/salidas de la capa; esto

es, sobre las dimensiones n_{l-1} y/o n_l, mientras que la dimensión correspondiente al lote simplemente se replica.

Desde el punto de vista de los operandos para el núcleo computacional GEMM de la etapa FP en (2), esto implica que $W^{(l)}$ se distribuye sobre los procesos por bloques de filas, mientras que $A^{(l-1)}$ queda replicada. Cada proceso entonces calcula el producto entre su bloque fila local de $W^{(l)}$ y la copia completa de $A^{(l-1)}$ para obtener un bloque fila del resultado $Z^{(l)}$. Por ejemplo, con $P = 4$ procesos, el particionado en el esquema superior de la Figura 2, correspondiente a

$$\begin{aligned} Z^{(l)} &= W^{(l)} A^{(l-1)} + B^{(l)} \equiv \\ \left[\begin{array}{c} Z_1 \\ \hline Z_2 \\ \hline Z_3 \\ \hline Z_4 \end{array}\right] &= \left[\begin{array}{c} W_1 \\ \hline W_2 \\ \hline W_3 \\ \hline W_4 \end{array}\right] A + \left[\begin{array}{c} B_1 \\ \hline B_2 \\ \hline B_3 \\ \hline B_4 \end{array}\right], \end{aligned} \quad (5)$$

ilustra que el proceso p calcula $Z_p = W_p A^{(l-1)} + B_p$, para $p = 1, 2, 3, 4$. Sin embargo, en preparación para la siguiente capa de la etapa FP, $A^{(l)} = \sigma(Z^{(l)})$ tiene que quedar replicada sobre los P procesos. En principio, esto se puede conseguir a través de una comunicación de tipo Allgather [13].

A continuación, en la (sub)etapa BP-GC, ambos operandos de entrada a la GEMM en (3), $W^{(l)}$ and $G^{(l+1)}$, están distribuidos entre los procesos por bloques de filas. Consideremos pues el particionado de (3) representado en el esquema en la mitad de la Figura 2, con $P = 4$. El correspondiente núcleo computacional GEMM puede entonces realizarse como

$$\begin{aligned} G^{(l)} &= (W^{(l+1)})^T G^{(l+1)} \\ &= (W_1^T \cdot G_1 + W_2^T \cdot G_2 + \\ &\quad W_3^T \cdot G_3 + W_4^T \cdot G_4), \end{aligned} \quad (6)$$

donde el proceso p es responsable de calcular la actualización parcial $W_p^T \cdot G_p$, para $p = 1, 2, 3, 4$. En principio, esto requiere una comunicación de tipo Allreduce [13] para acumular estas contribuciones parciales en el resultado global $G^{(l)}$. Al final de esta reducción, cada proceso p almacena una copia completa de $G^{(l)}$ si bien, para la "siguiente" capa $(l-1)$, realmente solo necesita el correspondiente bloque fila p.

Finalmente, en la (sub)etapa BP-WU, tanto $G^{(l)}$ como $A^{(l-1)}$ in (4) están replicados en todos los procesos. Así pues, cada proceso puede calcular la parte local de la actualización a $W^{(l)}$ sin ninguna comunicación adicional. El esquema interior en la Figura 2 muestra que, para P=4 procesos, obtenemos el particionado

$$\begin{aligned} W^{(l)} &= W^{(l)} - \eta G^{(l)} (A^{(l-1)})^T \equiv \\ \left[\begin{array}{c} W_1 \\ \hline W_2 \\ \hline W_3 \\ \hline W_4 \end{array}\right] &= \left[\begin{array}{c} W_1 \\ \hline W_2 \\ \hline W_3 \\ \hline W_4 \end{array}\right] - \eta \left[\begin{array}{c} G_1 \\ \hline G_2 \\ \hline G_3 \\ \hline G_4 \end{array}\right] A^T. \end{aligned} \quad (7)$$

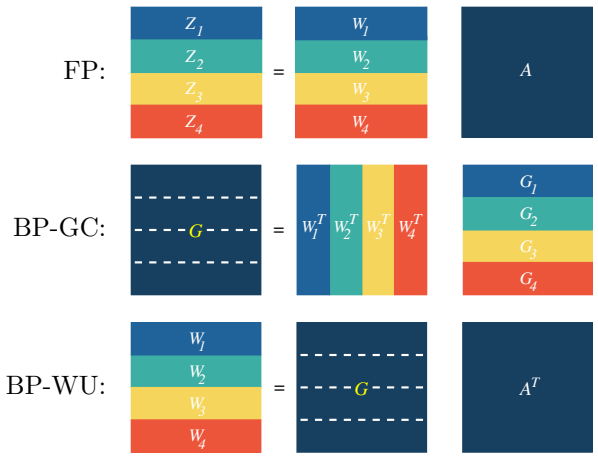


Fig. 2: Distribución de la carga en un esquema de paralelismo de modelo con $P = 4$ procesos. Cada bloque con color diferente se asigna a un proceso distinto. La matriz de pesos W se distribuye sobre todos los procesos. Las matrices de un único color se replican sobre todos los procesos.

En consecuencia, esta operación puede desarrollarse con el proceso p calculando $W_p = W_p - \eta\, G_p (A^{(l-1)})^T$, para $p = 1, 2, 3, 4$, mediante el bloque fila local de la matriz replicada $G^{(l)}$ y la copia completa de $A^{(l-1)}$.

El aprovechamiento del paralelismo de modelo en capas convolucionales sigue la misma aproximación que en las capas FC. En particular, en una capa de ese tipo es posible paralelizar la aplicación de una convolución asignando las operaciones correspondiente a un grupo de canales (*kernels* o filtros) de entrada a cada proceso. Las necesidades de comunicación entre procesos, para una capa de tipo convolucional, son las mismas que se han presentado para la capa FC.

Para cerrar esta exposición, cabe indicar que los datos que se comunican al final de la etapas FP (Allgather) y BP-GC (Allreduce) presentan las dimensiones del operando C en la Tabla I.

IV. Simulador de Paralelismo de Modelo

Para el estudio del paralelismo de modelo, hemos desarrollado el simulador Simulador de Paralelismo de Modelo (SPM) que integra las principales operaciones computacionales y de comunicación que aparecen en la secuencia de etapas FP, BP-GC y GP-WU presentes en el proceso de entrenamiento distribuido, tal y como se describe en las Secciones II y III.

En particular, nuestro simulador SPM lanza un proceso MPI por cada nodo del cluster, que se encarga de realizar localmente las correspondientes partes de las operaciones *im2col* y GEMM que aparecen en las capas de tipo FC y Conv durante el entrenamiento. Estas operaciones se realizan en paralelo usando una implementación multihebra propia de *im2col* y llamadas a la rutina apropiada a través de la interfaz de BLAS.

Durante el procesamiento de la capa l en las etapas

TABLA II: Parámetros del simulador SPM.

Parámetro	RN
L	#Capas de la RN
n_{l-1}	#Entradas capa l (FC)
n_l	#Salidas capa l (FC)
h_{l-1}	Altura de la entrada capa l (Conv)
h_l	Altura de la salida capa l (Conv)
w_{l-1}	Anchura de la entrada capa l (Conv)
w_l	Anchura de la salida capa l (Conv)
k_l^w	Anchura del filtro capa l (Conv)
k_l^h	Altura del filtro capa l (Conv)
c_l	#Filtros (canales) capa l (Conv)
b	Tamaño del lote

FP y BP, el simulador SPM solo utiliza los primeros $P_l = [n_l/d_{\text{FC}}]$ or $P_l = [c_l/d_{\text{Conv}}]$ procesos, en función de si la capa es de tipo FC o Conv, respectivamente. En cuanto a las comunicaciones, los intercambios de datos al final de capa capa de la etapa FP (Allgather) y justo después de cada capa de la (sub)etapa BP-GC (Allreduce) se realizan utilizando las correspondientes primitivas de comunicación colectiva de MPI.

Dado que el coste de entrenamiento de las RNs profundas está, en gran medida, dominado por los costes de las etapas FC y Conv, podemos esperar que el simulador SPM presente un comportamiento paralelo asintótico similar al de un entorno de entrenamiento de RNs que explotara el paralelismo de modelo.

A. Redes Neuronales

El simulador SPM está parametrizado para permitir la configuración de las capas FC y Conv de cualquier perceptron multicapa o red neuronal convolucional. La lista completa de parámetros que pueden ajustarse en el simulador SPM se muestra en la Tabla II.

B. Redes Neuronales

Las RNs convolucionales están caracterizadas por la presencia de capas Conv entrelazadas con capas de agrupación (*pooling*), y seguidas por una o más capas FC, responsables de obtener la respuesta (salida) de la RN. La Tabla III muestra el número y tipo de capas para cada cuatro modelos distintos de redes convolucionales.[1] Con el propósito de exponer la compleja estructura de estos modelos de RN, la Tabla IV detalla la arquitectura completa por capas de AlexNet, que está compuesta de cinco capas Conv, tres de condensación y otras tres capas finales de tipo FC. Las otras tres RNs, presentan en general un número de capas superior con incluso más neuronas por capa.

V. Evaluación del Rendimiento

En esta sección, ilustramos el funcionamiento del simulador SPM mediante dos RNs profundas conocidas como AlexNet y VGG16. En primer lugar, se describe la plataforma donde se han ejecutado las pruebas, y posteriormente se analizan los resultados obtenidos.

[1] La especificación de estos modelos se ha obtenido del banco de pruebas de Tensorflow [14].

TABLA III: Tipo y número de capas de las RNs convolucionales.

Modelo	FC	Conv	Pool	Total
AlexNet	3	5	3	11
Inception v3	1	94	14	109
ResNet-50 v2	1	53	1	55
VGG16	3	13	5	21

TABLA IV: Especificación de AlexNet.

Capa (l)	Tipo	Neuronas ($h_l \times w_l \times c_l$)	Filtros ($c_l \times h'_l \times w'_l \times c_{l-1}$)
0	Input	$224 \times 224 \times 3$	–
1	Conv	$55 \times 55 \times 64$	$64 \times 11 \times 11 \times 3$
–	MaxPool	–	–
2	Conv	$27 \times 27 \times 192$	$192 \times 5 \times 5 \times 64$
–	MaxPool	–	–
3	Conv	$13 \times 13 \times 384$	$384 \times 3 \times 3 \times 192$
4	Conv	$13 \times 13 \times 384$	$384 \times 3 \times 3 \times 384$
5	Conv	$13 \times 13 \times 256$	$256 \times 3 \times 3 \times 384$
–	MaxPool	–	–
6	FC	4,096	–
7	FC	4,096	–
8	FC	#Clases	–

A. Plataforma

En este trabajo, las RNs seleccionadas se han configurado sobre el simulador SPM, que se ha ejecutado sobre el cluster WST. Esta plataforma está compuesta por 16 nodos, cada uno equipado con un procesador Intel Xeon E5645 (Westmere) y 48 GB de memoria RAM DDR3. Los nodos de este cluster están conectados mediante un conmutador Mellanox QDR Infiniband.

La Tabla V recoge las especificaciones principales del cluster. El rendimiento máximo teórico (pico) por núcleo se ofrece en términos de operaciones en coma flotante FP32 (esto es, simple precisión); el ancho de banda máximo se calcula multiplicando la anchura del bus a memoria por la velocidad del reloj de la RAM y el número de canales de memoria.

TABLA V: Arquitectura del cluster WST.

Parámetro	WST
Modelo de procesador (Intel Xeon)	E5645
Productividad FP32 máxima (flops/ciclo)	8
Frecuencia (GHz)	2,4
#Núcleos	12
Rendimiento pico FP32 (GFLOPS)	115,2
Anchura bus memoria (bytes)	8
Frecuencia de reloj RAM (GHz)	1,333
Canales de memoria	3
Ancho de banda a RAM máximo (GBytes/s)	32
Memoria DDR3 (GBytes)	48
Red de interconexión (Mellanox Infiniband)	QDR
Capacidad de los enlaces (Gbps)	25,8
Latencia máxima de los enlaces (μs)	1,4

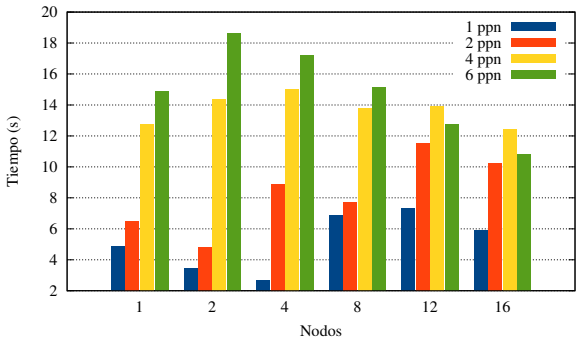


Fig. 3: Tiempo de ejecución del modelo AlexNet en WST con distintos niveles de paralelismo.

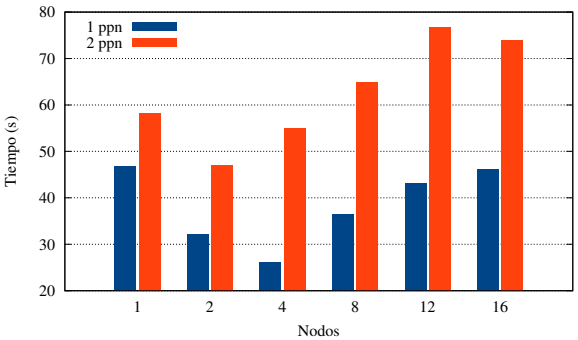


Fig. 4: Tiempo de ejecución del modelo VGG16 en WST con distintos niveles de paralelismo.

La biblioteca utilizada en WST que proporciona la implementación optimizada del núcleo computacional GEMM es Intel 2019 MKL. La biblioteca de comunicación que aprovecha las conexiones de la red Infiniband QDR en esta plataforma es MPICH v3.3.

B. Resultados

Para analizar el rendimiento del paralelismo de modelo en las RN, se han tenido en cuenta dos niveles de paralelismo: nivel de proceso (MPI) y nivel de hilo (OpenMP). El primer nivel permite dividir el modelo de la RN entre el número de procesos que se ejecutan mientras que el segundo nivel permite acelerar las operaciones que realiza cada proceso. Por lo tanto, cómo se dividen los recursos disponibles en esos dos niveles determina el rendimiento. Se han realizado combinaciones para 1, 2, 4, 8, 12 y 16 nodos donde se han ejecutado 1, 2, 4 y 6 procesos MPI por nodo para AlexNet, y 1 y 2 procesos MPI por nodo para VGG16, debido a la cantidad de memoria que requiere cada modelo. Una vez establecidos los procesos, se han generado tantos hilos de OpenMP por proceso como fuera necesario para asegurar que hay un hilo por núcleo en el nodo.

Las Figuras 3 y 4 muestran el tiempo total de ejecución para AlexNet y VGG16, respectivamente. Estos tiempos incluyen tanto el cómputo como las comunicaciones de las etapas FP y BP. La primera característica a destacar es que el paralelismo de modelo no escala cuando se añaden más recursos computacionales. De hecho, ni tan siquiera se obtiene el mejor resultado con la configuración que más recursos utiliza. Para ambos modelos, utilizar solamente 4 procesos en 4 nodos y 12 hilos de OpenMP en cada nodo obtiene el mejor rendimiento. Esto se debe, principalmente, a dos motivos. Por un lado, aunque se esté diviendo el modelo entre los procesos y por tanto cada uno tenga menor carga, puede darse la situación de que la cantidad de trabajo por proceso sea insuficiente. Por ejemplo, los tamaños de las matrices de la operación GEMM pueden no suponer una carga aceptable para obtener el rendimiento adecuado. Por otro lado, el incremento en el número de procesos ralentiza las comunicaciones colectivas y, por tanto, introduce un sobrecoste considerable.

Otro aspecto interesante a destacar es que el mejor rendimiento para cada combinación de número de nodos se consigue al asignar solamente un proceso por nodo, puesto que de esta forma se explota mejor el paralelismo a este nivel.

Si comparamos la mejor combinación con las otras que utilizan 4 procesos (1 nodo con 4 procesos por nodo, o 2 nodos con 2 procesos por nodo) vemos que el hecho de agrupar procesos en nodos, para evitar comunicaciones por red, reduce el rendimiento al no poder explotar el paralelismo a nivel de nodo, puesto que el número de hilos se reduce para no producir una asignación de más de un hilo por core (oversubscription).

Por lo tanto, en este tipo de modelos de RN, el paralelismo de modelo ofrece una escalabilidad muy limitada. Además, la importancia que tiene el paralelismo a nivel de nodo por las operaciones *im2col* y GEMM puede resultar en una pérdida de rendimiento al dividir el modelo en partes muy pequeñas.

VI. Conclusiones

En este trabajo se ha presentado un simulador de paralelismo de modelo para RNs que imita el patrón de cálculo y cómputo de esta solución para modelos de RNs distribuidos. Este simulador permite entender y optimizar el comportamiento de los entornos de programación para RNs sin la intervención de los datos para entrenar/inferir el modelo ni de los entornos de programación.

Se ha analizado el rendimiento de dos modelos conocidos de redes convolucionales para distintas configuraciones de procesos e hilos en un clúster de 16 nodos. Los resultados muestran que para este tipo de paralelismo, al contrario que en el paralelismo de datos, no se produce un incremento de las prestaciones al utilizar más recursos. Asimismo, los experimentos muestran que el peso de las comunicaciones colectivas (que no se puede solapar con el cálculo) influye en el rendimiento total de la aplicación.

Agradecimientos

Este trabajo ha sido financiado por los proyectos TIN2017-82972-R and RTI2018-098156-B-C51 del Ministerio de Ciencia, Innovación y Universidades. Manuel F. Dolz está financiado por el Plan GenT CDEIGENT/2018/014 de la *Generalitat Valenciana*.

Referencias

[1] Li Deng et al., "Recent advances in deep learning for speech research at Microsoft," in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, May 2013, pp. 8604–8608.

[2] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, USA, 2012, NIPS'12, pp. 1097–1105, Curran Associates Inc.

[3] Jiajun Zhang and Chengqing Zong, "Deep neural networks in machine translation: An overview," *IEEE Intelligent Systems*, vol. 30, no. 5, pp. 16–25, Sep. 2015.

[4] Vivienne Sze et al., "Efficient processing of deep neural networks: A tutorial and survey," *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, Dec 2017.

[5] Maryam M. Najafabadi et al., "Deep learning applications and challenges in big data analytics," *Journal of Big Data*, vol. 2, no. 1, pp. 1, Feb 2015.

[6] Tal Ben-Nun and Torsten Hoefler, "Demystifying parallel and distributed deep learning: An in-depth concurrency analysis," *CoRR*, vol. abs/1802.09941, 2018.

[7] Samira Pouyanfar et al., "A survey on deep learning: Algorithms, techniques, and applications," *ACM Comput. Surv.*, vol. 51, no. 5, pp. 92:1–92:36, Sept. 2018.

[8] Catherine F. Higham and Desmond J. Higham, "Deep learning: An introduction for applied mathematicians," 2018, arXiv:1801.05894.

[9] Kumar Chellapilla, Sidd Puri, and Patrice Simard, "High performance convolutional neural networks for document processing," in *International Workshop on Frontiers in Handwriting Recognition*, 2006.

[10] Aravind Vasudevan, Andrew Anderson, and David Gregg, "Parallel multi channel convolution using general matrix multiplication," in *2017 IEEE 28th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, July 2017, pp. 19–24.

[11] Andrew Anderson et al., "Low-memory GEMM-based convolution algorithms for deep neural networks," *CoRR*, vol. abs/1709.03395, 2017.

[12] Mu Li et al., "Scaling distributed machine learning with the parameter server," in *Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation*, 2014, OSDI'14, pp. 583–598.

[13] Ernie Chan, Marcel Heimlich, Avi Purkayastha, and Robert van de Geijn, "Collective communication: Theory, practice, and experience," *Concurr. Comput.: Pract. Exper.*, vol. 19, no. 13, pp. 1749–1783, Sept. 2007.

[14] Google Inc., "Tensorflow benchmarks," .

Preparing and managing an HPC Cluster: Lessons learned

Eduardo José Gómez-Hernández [1] y José Manuel García [2]

Abstract— **From the 1990s, building systems able to reach the performance of supercomputers was a priority, mainly by its cost. In the early 2000s, processors started to reach a limit in the power, forcing a change in the paradigm. Our research group is focused on High-Performance Computing (HPC), and having our own cluster allow us to adapt it to our needs at any moment. We present how we built the current research group cluster, reusing most of the parts of the older one, the most important decisions done, and what we have learned during this year. This new configuration is running and has been used to reproduce research scenarios correctly with near 6000 jobs sent to the workload system.**

Keywords— **High-Performance Computing, Computer Cluster, Heterogeneous Computing**

I. Introduction & Motivation

From the early 1990s, there has been motivation for building systems able to reach the performance of traditional supercomputers with lower cost. With this idea, Networks of Workstations (NOW) [1] were developed to replace supercomputers for certain types of applications. The High-Performance Computing (HPC) race was starting, and some clusters (such as Beowulf Cluster [2]) became a good rival of supercomputers but costing a tenth of its competitors price.

In the consequent years (near 2000s), processors started to reach a limit in power density, a slowdown in performance improvements was coming. The thread-level parallelism broke this problem allowing the creation of new multi-core processors and increasing the throughput when the workload consists of independent applications [3] but making impossible the creation of supercomputers made from only one machine.

The HPC race continues, every 6 months the TOP500 list is updated with the fastest computers in the world. All of them are heterogeneous machines clustered together, such as the Beowulf Cluster. Currently, the largest supercomputers in the world are competing to break the exascale barrier in floating point operations ($10^{18} FLOPS$).

Besides the HPC race, the main purpose of the computing power given by these powerful machines enables the execution of complex applications such as simulations, mathematical resolutions, prime number calculus (essential for the cybersecurity), cryptocurrency market, machine learning, and so on. These systems are being used anywhere, from weather simulation, gaming servers of thousands of players, models able to classify complex images or predict diseases automatically, among many others.

Our research group is mainly focused on HPC and its applications. Having our own cluster allow us to adapt it to our needs at each moment. With the revolution of machine learning and deep learning, we had to add several machines with special hardware for this task.

This paper presents the main decisions made to build and maintain our research cluster, making it flexible enough to be able to execute jobs of near every type, and providing a good replication environment for other researches.

We improved the previous GNU/Linux computer cluster with a new operating system, modularized the important software, automatized the tedious tasks, and monitored the entire cluster. From the user feedback and the easy daily maintenance, we ended with a success.

The rest of the paper is organized as follows: Section II presents the process of the hardware selection and its installation. Section III describes the system management, from user administration and permissions to monitoring the whole system. Section IV introduces what we have learned during this experience. Section V defines other interesting approaches and useful information. Finally, Section VI draws conclusions.

II. Installing

A. Starting Point

Our research group started with clusters in 2006. They had several machines racked in a room, but with the years, they become obsolete. Therefore, in 2015, José Antonio Bernabé and Victoriano Montesinos improved it. The improvement was done by rebuilding it from the start. This time, the machines were organized like a hierarchical cluster, but all of them were connected to the public network, being able to connect any machine at any moment without crossing the head node. But, the login and batch systems were centralized into the head node.

The cluster worked fine for a couple of years, but at the end of 2017, maintenance was too complicated. Powering the cluster after a shutdown (o a power outage) was painful, most of the machines required manual operation. Installing new software, was impossible without breaking any other.

We put up with the state of the cluster until July 2018, when we decided to shut down the cluster to rethink its organization. Then, in September 2018, the project to rebuild the GACOP's Cluster started.

[1]Computer Engineering Department, University of Murcia email: eduardojose.gomez@um.es

[2]Computer Engineering Department, University of Murcia email: jmgarcia@um.es

TABLA I
MACHINES MAIN SPECIFICATION SPLIT BY ITS PURPOSE IN THE CLUSTER.

Head Nodes

Machine	CPU	Memory	Accelerators
Main Head	Intel i7-8700 3.2GHz	8 GiB DDR4 2667 MHz	None
Backup Head	N/A	N/A	N/A

CPU Compute Nodes

Machine	CPU	Memory	Accelerators
Mendel	2xIntel Xeon E5-2698 v4 2.2GHz	4x32 GiB DDR4 2400MHz	None
Pacioli	2xIntel Xeon E5-2650 v2 2.60GHz	4x8 GiB DDR4 1333MHz	3xXeon Phi KNC x100
Lejeune	Intel Xeon Phi 7210 1.4GHz	4x32 GiB DDR4 2133MHz 16 GiB MCDRAM	None
Pascal	Intel Xeon Phi 7250 1.4GHz	6x32 GiB DDR4 2400MHz 16 GiB MCDRAM	None

GPU Compute Nodes

Machine	CPU	Memory	Accelerators
Watt	2xIntel Xeon E5-2603 v3 1.6GHz	4x16 GiB DIMM 2133MHz	Nvidia GTX 1080 (GP104)
Ampere	Intel Xeon E5602 2.4GHz	4x4 GiB DIMM 1066MHz	Nvidia GTX 980 (GK110GL)
Nikola	Intel Xeon E5602 2.4GHz	4x4 GiB DIMM 1066MHz	Nvidia Tesla K40c (GK110BGL)
Volta	2xAMD Opteron 6134	4x4 GiB DDR3 1333MHz	Nvidia Tesla K20c (GK110GL)

B. Avaliable Hardware

One of the requirements of the every new cluster is to reuse all the useful parts from the previous ones. Therefore it is necessary to evaluate the existing hardware to find if it meets our requirements. In our case, this hardware is compound by 9 high-performance machines, and a pair of network switches and KVMs (Keyboard Video Mouse switch).

In these machines, there are multiple devices such as Nvidia's GPUs, Intel's Scalable Xeon CPUs, AMD's Magny Cours CPUs, and Intel's Xeon Phi CoProcessors, among others. Depending on the needs it is possible to add or remove this hardware from the cluster, therefore, knowing what usage will it have is crucial before building it.

C. Purpose

This cluster is used mainly by professors, researchers, and students during their bachelor's, master's, and Ph.D. thesis. The cluster must execute jobs sent by users with some privileges. Since many of them require time measurement, the mutual exclusion must be guaranteed, but not only for the CPU, also the disk access, RAM, etc. Therefore, each machine will have the needed data in its own disk.

With all of the above in mind, we have physically configured the machines with the characteristics observable in Table I.

D. Networking

Since data is stored at each machine independently, the machines must be interconnected to exchange data, also to be able to work together as a single machine. For isolation and security, all compute nodes will remain connected by an internal network, and only the head node will have access to the outside network.

All the machines have, at least, one gigabit ethernet card, therefore we will use two gigabit network switches for communications. This is not ideal, and an InfiniBand network could improve performance. But, in our case, the machines normally are executed in mutual exclusion, meaning that the communications between machines should not affect performance.

Using this network configuration alongside with each machine specification, the composition is visible at the Figure 1

E. Operating System

In a computer cluster, the main features required to the operating system are performance and security. An operating system has to provide an abstraction to the hardware, manage the resources, provide security, and many other things.

There are several operating systems that can be used: GNU/Linux, Solaris, BSD, Windows Server, etc. But currently, the High-Performance market is replete with custom GNU/Linux systems. The main reason for being the main operating system used is its security, performance, and code availability, allowing anyone to report and fix any bug or security flaw.

Also, inside the GNU/Linux operating system, there are a lot of distributions. A distribution is a specific version of GNU/Linux with a lot of preconfigured settings and built-in software, and the most known one in the HPC environment is CentOS Linux, based in Red Hat Enterprise Linux. Another great choice is Scientific Linux (supported by Fermilab), but it is recently getting into its end of life. THeir authors will continue to support the latest versions, but they are migrating to CentOS.

Our selection was CentOS Linux, because it is a very stable distribution with big community support, and most of the scientific applications and libraries support it. Also, one of its objectives is to make a reproducible platform, which nowadays, is necessary for research.

F. Nvidia GPUs

The GPGPU (General Purpose GPU) programming has become a requirement in some applications to make them run at reasonable times, like Deep

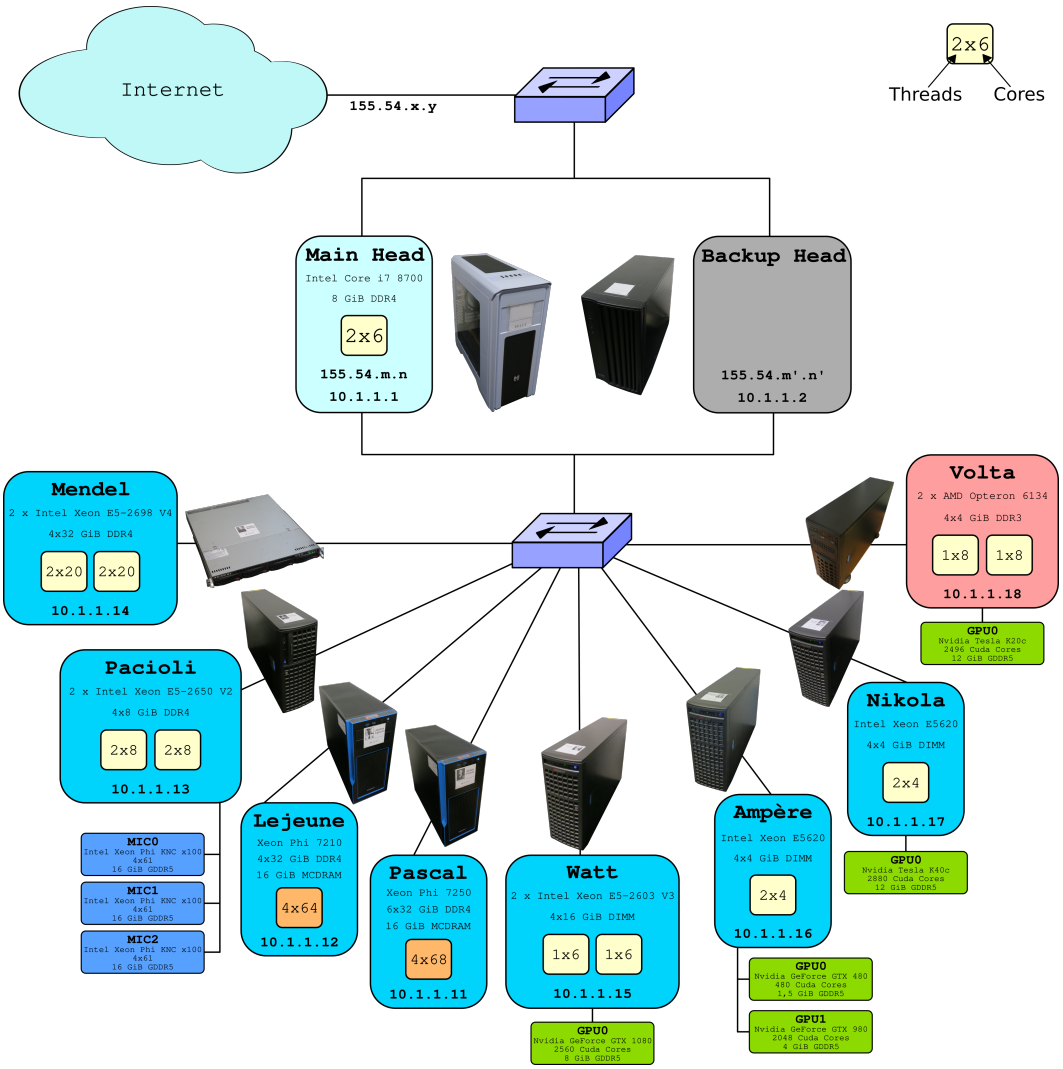


Fig. 1. Overview of the cluster architecture.

Learning. It is a powerful accelerator for SIMD (Single Instruction Multiple Data) paradigms. Nowadays there are three main GPU designer: Nvidia, AMD/ATI and Intel. Each designer has its own installation process, even could differ between different GPUs from the same designer, and its own language or programming methodology. For example, the main programming option for Nvidia is CUDA, but OpenCL is available for all of them (with less performance).

Normally, installing a GPU is not very complicated. To get all the computing power it can offer, it is necessary to download the proprietary driver from the designer's webpage, and install it. Sometimes, it may conflict with the open source driver, if it was previously installed. Lastly, if the device requires a special programming environment, such as CUDA or OpenCL, it needs to be installed aside.

G. Xeon Phi KNC

Intel Xeon Phi KNC (or Xeon Phi x100) is a coprocessor made by Intel using the MIC (Many Integrated Core) architecture [4]. It was recently deprecated [5], but it continues to be very useful for doing some jobs and must not be disregarded.

For GPUs, normally you install a driver and an SDK and is almost done, but Xeon Phi Coprocessors are a little bit different. The Intel MPSS (Intel Manycore Platform Software Stack) includes the driver and most of the tools needed to manage a Xeon Phi Coprocessor. It is currently under long term support, but not compatible with the latest versions of RHEL or Centos (7.5 and 7.6). Fortunately, Jan Just Keijser continues patching it from the Intel sources [6].

To start creating binaries that can use the coprocessor, Intel Parallel Studio is required, specifically the last version from 2017. But, if another Intel Parallel Studio or Intel Compiler is installed, a conflict could appear. An easy way to change between one compiler to other must exists for the cluster's users. In our case, we have two versions (2017 and 2019) packed inside modules, therefore the user can switch from one to other like any other existing module. In addition, we have created wrappers for most used applications from the Intel Parallel Studio to change automatically to the 2017 version without noticing the user.

Lastly, to launch jobs to the Xeon Phi, we are using different queues, one for each coprocessor, and

another for the host machine, allowing the native execution or the offloading one.

III. Management

A. Users and Directories

Users have to log in using the head node, but all machines need to know which user is. To afford it there are various user management services. From the most complex one, LDAP, to the simples, NIS. The cluster is not particularly big, therefore something too complex like LDAP is not needed. Keras is also a valuable candidate, but similar to LDAP, it is to complex to handle medium-size cluster. NIS makes exactly what we need, share users and groups between machines.

Then, when launching a job, due to the resources mutual exclusion, the program, and the data must be at the desired machine to be able to execute it. This can be achieved with the help of some programs. First, NFS allows mounting a directory from another machine easily. Then AutoFS can manage this for us, unmounting the directory when is not needed. Lastly, using a custom script, the machines' directories are linked to the home directory of each user. Since not all users have access to all machines, it is necessary to restrict access to only the ones he needs, a simple way to do it is using groups. If a user has the group "machine1", he will have a directory with name "machine1" linked to the NFS directory of that machine.

B. Batch System

One way to maintain mutual exclusion between users is to force them to use a batch system. A batch system (or workload system) manages access to the resources thought jobs using queues. There are many different policies to establish these queues. They can represent from a specific unit of the system to an entire system. Also, they can overlap themselves and using any kind of priority system order the execution of the jobs.

The most popular batch systems are Torque and SLURM. Our users are used to SLURM, but there are new users who come from another cluster in the university which uses Torque. To accomplish this, SLURM has a compatibility layer with Torque, allowing users to use the system they prefer without knowing which one is behind it.

C. Software

Maintain software in a cluster is a very tedious task. They can have security flaws, incompatibilities, or simply a user needs a specific version for any reason. When using operating systems of the Red Hat Server family, you end with a very old compiler (at least until RHEL 7), and all libraries depend on it. But, to support new features like different accelerators, or new platforms, it is necessary to use the new versions. Some time ago, the C++ ABI changed, and all software and library for Red Hat Server family use the old ABI, making a headache every time a user wants new software.

Unfortunately, there is not an easy system to do this. There are some approximations like easybuild, but they are not a swiss knife. There are always situations where you need to manually install some specific version of specific software.

There is another approximation called Environment Modules, a simple package that allows change user's environment variables through rules in a modulefile. In this case, a module is not restricted to anything specific, if it is possible to use a software modifying only environment variables, then is possible to use Environment Modules.

In our case, we have a special directory for modules, synced between all machines. We have several versions of GCC and Intel compilers, and we are planning to add LLVM. Also, we have a lot of deep learning modules for python, and some specific libraries and applications. To generate the modulefile for Intel compiler, we have used env2, a tool to convert environment variables from one scripting language to another.

D. UPS

When aiming to have a cluster powered during months, a power outage is a catastrophic problem. A power outage can corrupt information, stop jobs that could be running for months, or in the worst case, cause physical damage. Having some UPSs is a must when building a cluster.

Our most power hungry machine has a UPS only for it, and it depletes in some seconds, sometimes in 1 or 2 minutes. Therefore, we prefer saving data and prevent any kind of damage to the users' data shutting down the machines as fast as we can when detecting a power outage.

The current UPS take around 10-15 seconds to detect it and report to the head node. Then, the head node sends a signal to shut down each machine in parallel and it shutdowns itself. In these kinds of situations, we prefer integrity over availability.

E. Maintenance & Monitoring

Maintaining a cluster requires maintaining two different things, software, and hardware. The hardware must be secured, avoiding high temperatures and any kind of disaster, and the software must be secure and fail-safe. We have our cluster located in a specific room only for it, with a constant ambient temperature about 19-21 °C. Also, only authorized people have access to it, even some administrators can only access remotely.

On the other hand, we have blocked any petition not attending port 22, and only active users can log in. We recommend an RSA key authentication, but password login is allowed with some length restrictions. If an unauthorized person enters into a user account, he only has access to that user data and permissions. Any system file can only be modified by its own user. And the root account is password

protected and cannot be logged from outside.

There are some monitoring services to check that all is running fine and there is nothing strange. A monitoring service will check all the inserted rules at certain timed moments, and if something is not working as expected, it will set an alert taking actions or informing the administrators. We have selected Nagios, a well-known monitoring system, highly configurable and easy to use, adding rules as we see it necessary. At this moment, we are checking availability, disk space, and users. Since machines are used through the workload system, only the admins and the workload system can be logged in.

F. Documentation

In a cluster environment, there are at least two different documentation reports. The first one is the administrator document, it contains all the installation process, emergency situations, hardware information, and any other relevant information necessary to make the cluster work from the start, or after an emergency situation. This document can be passed to one administrator to another to improve or update it over the years. It has to track all the different changes made to the cluster. The second document is for the users, it must contain anything the user has to know to use the cluster from the very beginning (connecting to the cluster for multiple operating systems) to launch jobs, load software, upload files, etc. Also, a brief overview of the physical architecture of the cluster usually helps.

There are many ways of doing these documents, but we have chosen a wiki for the administration document, and a manual for the second one. The wiki has been developed similar to a diary, specifying each day the work done, the issues found, steps to solve those issues, etc. The manual includes a lot of step by step sections to help the user to get into the usage of the cluster.

IV. Lessons Learned

We have deployed a CentOS 7.5 HPC cluster for researching and educational purposes. Getting all the experiences throughout its building and maintenance this year. We would like to show the main lessons we have learned during this period.

Building a computer cluster is not a trivial task, after working with a couple of machines, scaling it may appear to be very simple, but in fact, it is not. There are always troubles with machine names, collisions between services, etc. In spite of the normal way to install a cluster is to prepare the head node and later a compute node and replicate it, it is necessary to never forget that there are more machines and each machine could be different.

Parodying the KISS principle (Keep It Simple, Stupid), for the users, the cluster must be simple to use and keep running, KISR (Keep It Simple and Running). Users normally do not matter how the cluster works, or the time needed to install new software or adapt the existing one to their necessities. Automatization is the key to making this view.

The security is very important, not only the allowed actions of a user, but also the outside attacks. First, to prevent the users to use unauthorized resources, we used a permission system using UNIX groups. Also, executing binaries from home directories (at the head node) is not allowed, this is not only for security, but it is also to prevent mistakes when running applications in local. Otherwise, the outside attacks are different, we have two main types of them: username and password cracking, and services attack. We receive around 4,000 invalid user logins each day, therefore we force the users to have passwords with at least some strength. Service attacks are different, they try to find a security flaw in a running service. In our case, only SSH is visible from outside (using iptable/firewalld), making it the only service available.

Good documentation is a time- and life-saver. At any moment, anything can happen and require manual intervention, and having any previous issue documented can save a lot of time. Also, when adding a new machine it is possible to repeat exactly the same steps as others, and only change the specific settings for that new machine, or hardware. Additionally, these documents can evolve, and get better with the time becoming a valuable piece of information in the future. Lastly, if a new administrator takes the cluster, he can review all the tasks done to the cluster, and improve any of them.

V. Related Work

From the creation of the Beowulf Cluster, a 16 Intel DX4 machines cluster [2], a lot of systems emerged. A lot of them with the same principle of low-cost High-Performance, but also clusters based on low power consumption. This movement allowed the creation of many systems with extravagant machines, and it is worth to mention some of them.

"Iridis-Pi" is a computer cluster built from 64 Raspberry Pi B computers [7]. Its main objective was to work as a single machine for education. Also, videogames consoles have been converted into a cluster. One of the most recent videogame consoles to be converted into a cluster is the PlayStation 3 [8]. Nowadays, smartphones are everywhere, therefore there are clusters made with them, like the "Droid-Cluster" [9].

We are in a heterogeneous era, where accelerators are essential in computing, hence many heterogeneous clusters with accelerators (such as GPUs and FPGAs) like Axel [10] and QP [11] were developed.

There are a lot of cluster administrators with their own clusters and sharing their experiences through the internet, and developing gold valuable documents for others. "Server World" [12] is a web page dedicated to making manuals configuring servers. Its main operating system is CentOS, but they have information for other systems. It is not limited to one configuration, there are manuals for a lot of different services, and also alternatives to the main ones.

Also, the "ADMIN Magazine" has two goods references when building a cluster [13], [14], they are a little bit old, but the fundamentals are the same.

Lastly, is worth to mention that the University of California (Irvine) has two large clusters running, and they are planning to deploy a new one. To know the requirements, they have a large report studying the utilization of the actual ones [15]. It is very interesting to understand how to evaluate a cluster and the main factors in deploying a new one.

VI. Conclusions

Building an HPC cluster is not an easy job, the system administrator has to teach the users how to use it, maintain the system up and running, install the necessary software and hardware, keep the system secure without allowing unauthorized people to use the cluster, or even preventing some users from using some parts of it, among many other tasks.

During this year, this new architecture of the cluster has been tested with the users, they are running several jobs from different environments. And some research scenarios have been reproduced correctly, getting, in most of the cases, the same results than the original authors. At the moment of this writing, near 6000 jobs have been run in the batch system.

We have found some things that can be improved to get better user experience and more flexibility. We will implement them the next time we have to rebuild the cluster for a major update. For example, we found that a shared directory is necessary for jobs with a lot of data, or when time is not a restriction.

Acknowledgements

This work was partially supported by the Spanish MCIU and AEI, as well as European Commission FEDER funds, under grant RTI2018-098156-B-C53. We acknowledge the excellent work done by José Antonio Bernabé and Victor Montesinos who developed the old cluster architecture and documents while they were doing a research internship supported by the University of Murcia. Also, Gonzalo Caparrós Laiz who is helping us in some administration tasks.

Referencias

[1] William Kramer, "Clustered workstations and their potential role as high speed compute processors," 1994.

[2] Thomas L. Sterling, Daniel Savarese, Donald J. Becker, John E. Dorband, Udaya A. Ranawake, and Charles V. Packer, "BEOWULF: A parallel workstation for scientific computation," in *Proceedings of the 1995 International Conference on Parallel Processing, Urbana-Champain, Illinois, USA, August 14-18, 1995. Volume I: Architecture.*, 1995, pp. 11–14.

[3] Antonio Gonzalez, "Trends in processor architecture," *CoRR*, vol. abs/1801.05215, 2018.

[4] Andrey Vladimirov, Ryo Asai, and Vladim Karpusenko, *Parallel Programming and Optimization with Intel Xeon Phi Coprocessors*, Colfax International, 2 edition, 2015.

[5] Intel, "Intel xeon phi x100 product family," `https://ark.intel.com/content/www/us/en/ark/products/series/92649/intel-xeon-phi-x100-product-family.html`.

[6] Jan Just Keijser, "Intel xeon phi (knc) mpss modules for centos 7.5 and 7.6," `https://www.nikhef.nl/~janjust/mpss/`.

[7] Simon J. Cox, James T. Cox, Richard P. Boardman, Steven J. Johnston, Mark Scott, and Neil S. OŔrien, "Iridis-pi: a low-cost, compact demonstration cluster," *Cluster Computing*, vol. 17, no. 2, pp. 349–358, 2013.

[8] Fernando Pardo and Jose A. Boluda, "Laboratorio de arquitecturas avanzadas con cell y playstation 3," *XXI Jornadas de Paralelismo*, 10 2010.

[9] Felix Büsching, Sebastian Schildt, and Lars C. Wolf, "Droidcluster: Towards smartphone cluster computing - the streets are paved with potential computer clusters.," in *ICDCS Workshops*. 2012, pp. 114–117, IEEE Computer Society.

[10] Kuen Hung Tsoi and Wayne Luk, "Axel: a heterogeneous cluster with fpgas and gpus," 01 2010, pp. 115–124.

[11] Michael Showerman, Jeremy Enos, Avneesh Pant, Volodymyr Kindratenko, Craig Steffen, Robert Pennington, and Wen mei Hwu, "Qp: A heterogeneous multi-accelerator cluster," in *In Proc. 10th LCI International Conference on High-Performance Clustered Computing - LCI'09, 2009. Memory reliability*, 03 2009.

[12] Server World, "Build network server," `https://www.server-world.info/en/`.

[13] Gavin W. Burris, "Setting up an hpc cluster," `http://www.admin-magazine.com/HPC/Articles/real_world_hpc_setting_up_an_hpc_cluster`.

[14] Jeff Layton, "The fundamentals of building an hpc cluster," `http://www.admin-magazine.com/HPC/Articles/Building-an-HPC-Cluster`.

[15] Harry Mangalam, "Hpc3 cluster planning stats," `http://moo.nac.uci.edu/~hjm/hpc/hpc3/HPC-Cluster-Stats.html`.

Evaluación de Prestaciones de Raspberry Pi para Entornos Fog Virtualizados

Javier Cimas, Carmen Carrión y M. Blanca Caminero[1]

Resumen— **El concepto de Internet de las Cosas (IoT) suele ir ligado al de computación en la nube, que ofrece recursos de almacenamiento y procesamiento virtualmente ilimitados para obtener información de los datos recogidos de las "cosas". No obstante, en los últimos años han surgido diversas aplicaciones IoT que no toleran bien las latencias introducidas por la comunicación con la nube, entre otros inconvenientes. Así surge el concepto de computación en la niebla (*Fog computing*), donde los recursos de computación y almacenamiento se acercan a las "cosas". Una de las posibles implementaciones del Fog se apoya en el uso de computadores monoplaca (SBC, *Single-Board Computer*), debido a su excelente relación precio/prestaciones/consumo.**

Por otro lado, hoy día las técnicas de virtualización ligera basadas en contenedores se usan ampliamente para ofrecer un cierto grado de aislamiento entre diversos servicios que comparten el mismo hardware. En este trabajo se evalúa el impacto de estas técnicas de virtualizacion (así como su necesaria orquestación) sobre las prestaciones obtenidas por diversos *benchmarks* sobre un ejemplo representativo de SBC, como es la Raspberry Pi. Esta información será de gran utilidad a la hora de planificar los recursos disponibles en un Fog basado en SBCs.

Palabras clave— **Computación Fog, computador monoplaca (SBC), virtualización, contenedores, benchmarks**

I. Introducción

ACTUALMENTE nos encontramos en una etapa caracterizada por el auge de las tecnologías de la información motivadas por la aparición del concepto de Internet de las Cosas (IoT) [1]. IoT es un paradigma que combina distintos aspectos y tecnologías: computación ubicua, protocolos de Internet, redes de sensores, tecnologías de comunicación y dispositivos embebidos.

El objeto inteligente es el elemento central de IoT, objetos de la vida diaria que recogen información ambiental e interactúan o controlan el mundo físico y están interconectados para intercambiar datos [1]. Actualmente podemos ver cómo estamos rodeados por diferentes dispositivos y aplicaciones que recogen grandes cantidades de datos. Desde los teléfonos inteligentes o los dispositivos *wearables*, en el ámbito de la informática personal o de usuario, hasta dispositivos que miden diversos parámetros ambientales (ruido, contaminación, temperatura, presencia, ...), útiles en el contexto de las ciudades inteligentes o de la agricultura de precisión, pasando por elementos que facilitan el seguimiento de activos en logística o la implantación de la Industria 4.0 [2].

Se trata, por tanto, de un amplio concepto que no sólo ha causado un gran impacto en la sociedad actual a partir de sus innumerables aplicaciones, sino que se encuentra en pleno desarrollo en sectores tales como la industria, la agricultura o la salud.

Las diversas aplicaciones de IoT se han apoyado tradicionalmente en los recursos ubicados en la nube (computación Cloud) para extraer valor a la potencialmente enorme cantidad de datos recopilados desde las "cosas". Como se detallará más adelante, los grandes proveedores de servicios en la nube ofrecen soporte para IoT, centrados en los aspectos básicos de ingesta, almacenamiento y análisis masivo de datos. Sin embargo, existen algunas aplicaciones donde esta aproximación no es viable, debido a las latencias que introduce la comunicación hacia/desde la nube, ni eficiente, debido al ancho de banda consumido en enviar todos los datos hacia la nube para ser procesados. Es ahí donde surge el concepto de computación en la niebla (o computacion Fog), donde los recursos de almacenamiento y procesamiento de los datos se acercan a las "cosas" donde se generan.

Es en el contexto de una arquitectura Fog donde se plantea el presente trabajo. En una arquitectura Fog existe capacidad de cómputo y almacenamiento más cercana a los objetos, más ubicua (sin perjuicio de que se complemente con recursos adicionales, potencialmente infinitos, en servidores Cloud). Existen diversas estrategias para dar soporte a esta capacidad, desde integrarla en los propios dispositivos de red [3] a establecer "micro-datacenters" cercanos a los objetos [4]. Otra posibilidad es aprovechar dispositivos de bajo coste, del tipo *Single-Board Computer* (SBC) que ofrecen una excelente relación precio/prestaciones. Uno de los dispositivos SBC más usado es la Raspberry Pi [5].

Otro de los aspectos a considerar es que idealmente la infraestructura Fog debería de ser capaz de soportar diversas aplicaciones IoT simultáneamente. Por ejemplo, en una fábrica inteligente podrían coexistir una aplicación IoT para el control ambiental en función del uso de las instalaciones, una aplicación de seguridad y control de presencia en ubicaciones sensibles, y otra aplicación para la monitorización del estado de salud de los trabajadores. Cada aplicación debería de ejecutarse de forma aislada del resto por motivos de seguridad, pero también en aras a proveer a cada una con la Calidad de Servicio (QoS) que requieran. Uno de los elementos que facilitan esta labor y que a priori introducen poca sobrecarga en los sistemas es el enfoque basado en virtualización ligera (contenedores) [6].

Así pues, el objetivo principal que se plantea en este trabajo es evaluar la sobrecarga que introduce

[1]Dpto. de Sistemas Informáticos, Universidad de Castilla-La Mancha, e-mail: javier.cimas@alu.uclm.es, carmen.carrion@uclm.es, mariablanca.caminero@uclm.es

la implantación de esquemas de virtualización ligera (contenedores) y su orquestación, sobre los limitados recursos que ofrece una Raspberry Pi. Se evaluará el impacto en tres dimensiones: capacidad de cálculo, de entrada/salida y de transferencia por la red. Esta información será de gran utilidad a la hora de desarrollar criterios para la gestión de los recursos en la arquitectura Fog distribuida.

II. Antecedentes

A. De las nubes a la niebla

El término Fog se usa en contraste con el bien conocido término "computación en la nube" (*Cloud Computing*). Mientras que en un paradigma Cloud toda la capacidad de almacenamiento y procesamiento se haya centralizada en centros de datos virtualizados, generalmente repartidos alrededor del mundo, el concepto de Fog acerca esa capacidad a los sistemas finales, en este caso, a las "cosas". De esta manera, se obtienen ventajas, sobre todo en cuanto a la reducción de latencias de comunicación, que pueden resultar muy sensibles en aplicaciones de tiempo real. Por otro lado, es posible usar más eficazmente el ancho de banda al transmitir hacia el Cloud sólo los datos relevantes o que necesitan almacenamiento definitivo.

Más concretamente, este trabajo se plantea en un contexto en el cual el Fog estará compuesto por dispositivos pequeños y económicos, del tipo *Single-Board Computer* (SBC), que puedan ser desplegados sobre áreas potencialmente extensas. El ejemplo más representativo de estos dispositivos es Raspberry Pi [5]. Inicialmente, estos dispositivos actúan como pasarelas que almacenan la información recogida por los sensores y la reenvían a un proveedor Cloud externo, para su posterior tratamiento. Aplicando la filosofía del paradigma Fog, estos dispositivos realizarán además algún tipo de procesamiento sobre los datos, incluyendo la posible toma de decisiones (por ejemplo, en temas de control de alarmas o situaciones de peligro).

En cuanto a las tecnologías Cloud y Fog, su evolución e inmersión en nuestra sociedad también ha sido muy completa. Hoy día, la adopción de la computación en la nube es muy amplia, tanto en el ámbito académico como en el ámbito empresarial, y tanto para grandes empresas como para PYMES. En el marco concreto de IoT, los grandes proveedores de servicios Cloud (Amazon, Microsoft, Google) ofrecen servicios específicos para ello [7][8][9]. Estos servicios generalmente consisten en recopilar y enviar datos a la nube, facilitando la carga y el análisis de esa información empleando otros servicios alojados en el mismo proveedor Cloud, ofreciendo por tanto un ecosistema completo e integrado. Su punto fuerte es su capacidad para la ingesta masiva de datos, desde dispositivos potencialmente distribuidos geográficamente a lo largo de todo el globo terráqueo. Sin embargo, en algunos casos de uso (como la Industria 4.0) puede que no sean la solución más adecuada dados los tiempos de respuesta que involucra el llevar la información hacia sus centros de datos. Cuando se trata de dar soporte a sistemas ciber-físicos, el tiempo de respuesta es crucial, y más aún en aplicaciones que tienen que ver con la seguridad física de las personas. Es por ello que surge el paradigma de computación Fog, donde el procesamiento de los datos se acerca a los dispositivos, tanto sensores (fuentes de datos) como actuadores (llevan a cabo acciones tras el análisis de los datos y la toma de decisiones). Hay que reseñar que los enfoques Fog y Cloud no son excluyentes. Hay servicios y aplicaciones cuyos requisitos no funcionales (tiempo de respuesta, por ejemplo) las hacen candidatas de implementarse en el Fog, mientras que a la vez otras aplicaciones pueden implementarse en el Cloud (por ejemplo, almacenamiento de datos históricos, análisis de tendencias a largo plazo, etc.), donde la capacidad de almacenamiento y procesamiento es prácticamente infinita.

Un ejemplo de aplicación del paradigma Fog aparece en [10], donde se propone una arquitectura Fog para un entorno de Industria 4.0, y se evalúa mediante simulación, haciendo hincapié en las posibilidades que ofrece la programación de los nodos Fog. Otro ejemplo es [11], donde se proponen diversas estrategias de distribución de tareas y de ubicación de máquinas virtuales entre nodos móviles, en el contexto de sistemas médicos ciber-físicos. En este caso, se asume que los dispositivos que componen el Fog tienen una cierta capacidad de proceso (del tipo de la que se encuentra disponible en un smartphone de gama alta) y tienen una alta tasa de movilidad.

En cuanto al uso de dispositivos del tipo SBC en el Fog, existen una serie de estudios preliminares acerca las prestaciones que pueden ofrecer al incorporarles entornos de virtualización ligera [12][13], que es necesaria a la hora de "compartimentar" el uso de los recursos por motivos de Calidad de Servicio y seguridad. Sin duda, la combinación de técnicas de planificación de recursos sobre dispositivos sencillos y baratos, con limitaciones de proceso y almacenamiento, junto con técnicas de virtualización ligera suponen una vía prometedora para explorar la implementación de un Fog sostenible, eficiente y seguro. Incipientes estudios, como las simulaciones presentadas en [14] en el contexto de fabricación inteligente, apuntan en esta dirección.

B. Aislando aplicaciones

Uno de los objetivos deseables en una arquitectura Fog es que la infraestructura desplegada sea capaz de soportar distintas aplicaciones simultáneamente. Un ejemplo puede ser combinar el control del movimiento en una fábrica con servicios de monitorización de las condiciones físicas de los trabajadores. Ambos tipos de servicios pueden compartir algunos datos obtenidos del entorno (por ejemplo, localización del personal), mientas que otros datos serán exclusivos de cada tipo de servicio (por ejemplo, parámetros físicos personales como temperatura o pulso cardiaco). Además, diferentes servicios tendrán diferentes requisitos no funcionales: algunos servicios implican

más tráfico de red que otros, habrá servicios con requisitos de tiempo real, etc. No hay que olvidar tampoco los problemas de seguridad que pueden darse al tener varias aplicaciones funcionando sobre una infraestructura compartida, que se tendrán muy en cuenta debido a la naturaleza de los datos que se transmiten. En base al tipo de dispositivos que compondrán el Fog y al requisito de compartimentación entre servicios, se propone el despliegue de un tipo de virtualización ligera basada en contenedores para la implementación de las distintas aplicaciones y servicios alojados en el Fog. Un contenedor consiste básicamente en empaquetar una aplicación junto con las librerías que necesita [15]. El uso de contenedores es una tendencia en auge para el despliegue ágil de aplicaciones basadas en microservicios y supone una alternativa ligera a la virtualización tradicional, que incluye un sistema operativo completo en cada máquina virtual. Estudios recientes empiezan a explorar la posibilidad del uso de contenedores en dispositivos del Fog, debido principalmente a su menor sobrecarga en dispositivos con recursos limitados [12][13][16][17][18]. En este trabajo se ha escogido Docker [19], debido a su masivo uso en entornos de virtualización ligera.

C. Gestión de los nodos Fog

Las aplicaciones que hacen uso de la computación en la niebla necesitan una infraestructura distribuida gestionada de forma inteligente. Así, pensando en un cluster de contenedores desplegados en varios dispositivos SBC, se hace necesaria una plataforma de orquestación capaz de proporcionar balaceo de carga, que gestione la comunicación entre los dispositivos, que monitorice el estado del sistema y dé soporte a tolerancia a fallos. En este trabajo se ha seleccionado Kubernetes (k8s) porque se está convirtiendo en el estándar de facto para la orquestación de contenedores [20].

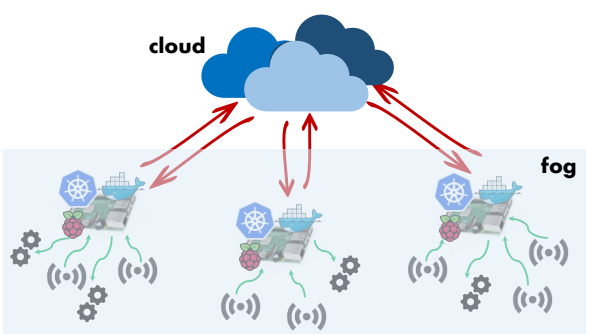


Fig. 1. Arquitectura Fog.

En la Figura 1, se ilustra el concepto de arquitectura Fog donde se enmarca este trabajo. Kubernetes se emplea para gestionar las aplicaciones desplegadas como contenedores de Docker en los dispositivos SBC que forman parte de la infraestructura Fog. Desde éstos, a su vez, las aplicaciones interaccionan de forma directa con los distintos sensores y actuadores que componen la base del Internet de las Cosas.

III. Entorno de pruebas

El objetivo del presente trabajo consiste en evaluar experimentalmente el impacto de la virtualización a nivel de sistema operativo y su orquestación sobre los nodos de cómputo de una arquitectura de computación Fog distribuida. En esta sección se detalla el entorno donde se han realizado las diversas pruebas de evaluación.

La Tabla I muestra las características hardware del computador monoplaca (*Single-Board Computer, SBC*) empleado como nodo en este análisis. Se trata de una Raspberry Pi 3 modelo B (RPi3).

TABLA I
Características del hardware usado.

CPU	Quad Core @1.2GHz ARMv8 Cortex-A53 (64Bit)
Chipset	Broadcom BCM2387
Memoria	1GB LP-DDR2 @400MHz
GPU	Broadcom VideoCore IV
Ethernet	10/100 Mb/s
Almacenamiento Flash	MicroSD
Connectivity USB	4 USB 2.0 Host
Kernel Version	4.14.98-v7+
OSType	linux

Mientras que Docker es una capa de abstracción software que gestiona el ciclo de vida de un contenedor y proporciona aislamiento entre ellos, Kubernetes gestiona los contenedores en un clúster dando soporte a tolerancia de fallos. Sin embargo, es difícil conocer el impacto que estas capas de abstracción tienen en las prestaciones finales, tanto de cara al usuario como desde el punto de vista del sistema. En este trabajo se presentan varios de los experimentos realizados en un entorno de pruebas controlado para medir la sobrecarga introducida por estos niveles de abstracción. Así, en este análisis se han considerado los siguientes escenarios de trabajo:

- *Entorno bare-metal*: En este escenario, en la Raspberry Pi se ha instalado el sistema operativo Raspbian GNU/Linux 9 (stretch) y el *benchmark* a probar.
- *Entorno Docker*: En cuanto a la virtualización, como tecnología de contenedores se ha empleado Docker versión 18.09.0. La imagen base de Docker usada para desplegar las aplicaciones de test es la imagen de Ubuntu 18.04.2 LTS.
- *Entorno Kubernetes*: Como orquestador de contenedores se ha instalado Kubernetes 1.14. En este punto cabe señalar que la unidad de gestión mínima de Kubernetes es el pod que alberga uno o más contenedores. En todas las pruebas realizadas se ha asignado un contenedor por pod.

La Figura 2 muestra el *testbed* desplegado. El cluster gestionado por Kubernetes consta de 3 nodos conectados entre sí (y hacia Internet) a través de una red WiFi. Dos de los nodos se han configurado como nodos de cómputo (*k8s workers*) y son seleccionables para ejecutar los pods. El nodo restante toma el rol de nodo maestro (*k8s master*) y se ha configurado para realizar en exclusiva las tareas de orquestación

del cluster. Uno de los nodos de cómputo, donde se van a ejecutar los *benchmarks*, se ha conectado a un medidor de energía externo (WattsUp PRO) para obtener datos sobre consumo energético.

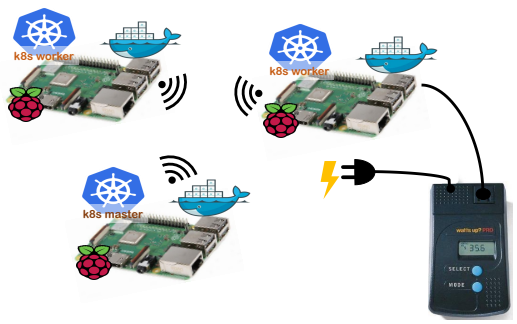


Fig. 2. *Testbed* desplegado.

Cabe destacar que en todas las pruebas se han empleado los mismos nodos físicos. Además, para cada entorno se ha hecho uso de un despliegue lo más limpio posible que evite la ejecución de procesos innecesarios en segundo plano, a fin de no afectar a las prestaciones medidas.

A. Benchmarks *sintéticos*

Los *benchmarks* sintéticos permiten generar distintos tipos de cargas para evaluar el rendimiento de un recurso concreto (CPU, red, I/O) del sistema bajo análisis. Este trabajo se centra en los recursos de cómputo (CPU), entrada/salida a disco y conexión de red. Se describen a continuación las aplicaciones y configuraciones usadas a tal fin en los experimentos realizados:

- **CPU**: Se ha empleado la aplicación multithread `pov-ray` (versión 3.7.0) para evaluar el impacto de la virtualización en el uso intensivo de CPU. `Pov-ray` es un software de trazado de rayos (*ray-tracing*) capaz de renderizar imágenes ultra realistas. Esta aplicación se ha usado en modo *benchmark* para evitar la salida gráfica y se ha ajustado en su configuración el parámetro *Quality* con un valor de 5. En el caso de Docker, se ha partido de una imagen base y se ha instalado la aplicación generando una nueva imagen con `pov-ray` instalado y configurado según se ha indicado.
- **Disk I/O**: Se ha usado `IOzone` como *benchmark* representativo de una aplicación que hace uso intensivo de lecturas y escrituras a fichero. Para la ejecución de esta prueba se ha deshabilitado el uso de memoria caché y se ha fijado el tamaño de los ficheros a 100MB. Además, se han realizado varias pruebas modificando el tamaño de bloque (4K, 512K y 16M).
  Para la evaluación de este *benchmark* se han considerado dos despliegues distintos. Docker Engine crea una serie de directorios dentro de *UnionFS* donde los datos del contenedor son almacenados de forma no persistente. Por otro lado, Docker facilita el uso de volúmenes para la persistencia de los datos una vez parado el contenedor. Puesto que las prestaciones en uno u otro caso pueden verse afectadas, en este trabajo se han analizado ambas opciones:
  Docker: Almacenamiento no persistente, a través del driver de almacenamiento `overlay2`.
  Docker-v: Almacenamiento persistente, a través del uso de un volumen montado en un directorio local.
- **Network I/O**: Para analizar el sobrecoste de Docker y Kubernetes sobre las comunicaciones de red se ha usado la herramienta *iperf* que permite obtener entre otros parámetros el ancho de banda máximo de la comunicación entre dos nodos. *Iperf* funciona en modo cliente-servidor por lo que se ha ejecutado el *benchmark* haciendo uso del protocolo TCP entre dos de los nodos del *testbed*.
  La instraestructura de red de un nodo del Fog es compartida por todos los contenedores. Además, todos los contenedores dentro de un pod de Kubernetes tienen asignada la misma dirección IP. Esto implica la necesidad de establecer mecanismos de traducción de direcciones IP para hacer viable la comunicación entre los servicios alojados en los distintos contenedores. Para medir el impacto de estas características, *iperf* se ha ejecutado en cada prueba durante 180 segundos haciendo uso tanto de la red WiFi 802.11n como de la red Ethernet cableada. En este último caso el medio está dedicado y se evitan así las fluctuaciones debidas a la compartición del medio de comunicación.

B. *Métricas*

Para evaluar la sobrecarga que supone tanto la virtualización de contenedores como la orquestación de los mismos en las Raspberry Pi sobre los subsistemas analizados con los distintos *benchmarks* se han monitorizado las métricas mostradas en la Tabla II. Además, para conocer el estado del sistema se ha medido el porcentaje de uso de CPU, el uso de memoria y la temperatura de la CPU del nodo Fog.

TABLA II
MÉTRICAS MONITORIZADAS POR APLICACIÓN.

Benchmark	Métrica
pov-ray	Tiempo ejecución (s)
IOzone	Operaciones de lectura y escritura (Kb/s)
iperf	Ancho de banda ($Mbits/s$)

Otra métrica importante monitorizada en los experimentos realizados es el consumo de energía del nodo, a través del medidor WattsUp PRO. Este medidor colocado entre el nodo Fog donde se realizan las pruebas y el enchufe de corriente (véase la Figura 2) proporciona de forma independiente los datos de la energía consumida por la Raspberry Pi durante el intervalo de monitorización definido previamente. WattsUp PRO almacena los datos en una memoria interna no volátil. La frecuencia de monitorización se

ha fijado a una muestra por segundo para todos los experimentos presentados en este trabajo.

Por último, antes de mostrar los resultados obtenidos en la siguiente sección, hay que mencionar que todas las pruebas se han ejecutado 4 veces y que los datos mostrados son los valores medios.

IV. Resultados

En esta sección se analizan los resultados experimentales obtenidos.

En primer lugar se presentan los resultados obtenidos al ejecutar la aplicación `pov-ray`, intensiva en CPU. Así, la Tabla III muestra los resultados medios obtenidos para las métricas bajo análisis para los tres entornos considerados (*bare-metal*, Docker y Kubernetes). Como se puede observar el coste de la virtualización con contenedores es insignificante pero la orquestación de los mismos introduce un incremento del 13 % en el tiempo de ejecución del *benchmark*, que se traduce como es lógico en un mayor consumo energético.

En cuanto a temperatura, destacar que en los tres entornos la temperatura máxima de la CPU se alcanza muy rápidamente (84^o Celsius), y se mantiene durante toda la prueba en esos valores. De esto se deduce que la temperatura, en esta prueba, es un factor limitante en cuanto al rendimiento alcanzable.

TABLA III
Resultados obtenidos con `pov-ray`.

Entorno	Tiempo ejecución (seg.)	Temperatura pico (°C)	Temperatura media (°C)	Consumo (Watts)	Potencia (J)
Bare metal	279.183	84	80.85	4.23	1178.11
Docker	279.523	84	81.05	4.19	1171,55
Kubernetes	316.978	84	82.16	4.28	1355.59

A continuación, se analizan las prestaciones de la entrada/salida en disco, realizadas mediante el *benchmark* `IOzone`. Las Figuras 3 y 4 muestran respectivamente las lecturas y escrituras realizadas por unidad de tiempo, para cada una de las cuatro configuraciones consideradas.

Los datos ofrecidos por la prueba de lectura de disco (Figura 3) no muestran diferencia alguna apreciable entre las cuatro configuraciones. El uso de Docker y Kubernetes se refleja en las pruebas de escritura, especialmente al incrementar el tamaño de los bloques (Figura 4). Hay que resaltar el impacto positivo en el rendimiento del uso de volúmenes en Docker. En particular, en el caso de las escrituras a disco, la configuracion de Docker con volúmenes es prácticamente idéntica a la *bare-metal*. Esto es consistente con la documentación de Docker, donde se recomienda el uso de volúmenes para almacenamiento por hacer un *by-pass* del controlador de almacenamiento de Docker [21].

Por último, en cuanto al análisis de sobrecoste sobre la infraestructura de red, la Tabla IV muestra los resultados obtenidos al ejecutar el *benchmark* `iperf`. En este caso, las pruebas se han llevado a cabo sobre dos configuraciones de red. Por un lado, se presentan los resultados cuando los dos extremos de la

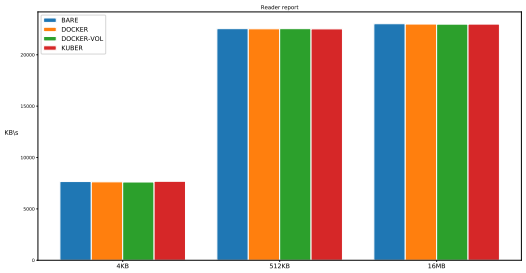


Fig. 3. Lecturas de disco con IOzone.

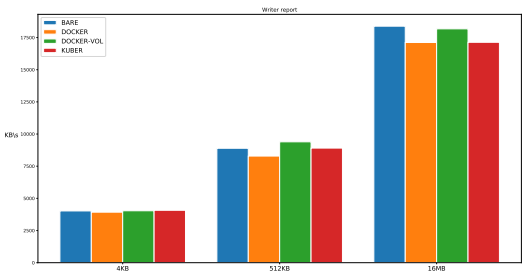


Fig. 4. Escrituras a disco con IOzone.

comunicación se conectan al mismo segmento de red FastEthernet dedicada. Esto da una idea de la sobrecarga de red introducida por el software, al ser un entorno ideal. Por otro lado, se incluyen también algunos datos obtenidos cuando el canal de comunicación es una red inalámbrica (un caso más cercano a la realidad). Para disponer de un entorno controlable para los experimentos, se ha empleado un punto de acceso dedicado a este fin.

En la Tabla IV se detallan tanto los valores medios como las desviaciones típicas obtenidas tras realizar una serie de 5 pruebas en cada entorno. Como es natural, las prestaciones obtenidas en el *testbed* sobre red cableada son mayores y más estables que en el entorno WiFi. En el caso de la red cableada, las prestaciones obtenidas con Docker son muy similares a las del despliegue *bare-metal*. Kubernetes sí que implica una pequeña penalización en el ancho de banda máximo observado (inferior al 4 %). Cuando se emplea conexión mediante la red WiFi, la tendencia es la misma, aunque la sobrecarga introducida por Kubernetes es significativamente mayor (un decremento del 17 % en el ancho de banda máximo, frente a un 2 % de penalización por el uso de Docker). La variabilidad es el doble de la observada en los entornos de Docker y *bare-metal*.

TABLA IV
Resultados obtenidos con `iperf`.

Entorno	Red cableada		WiFi	
	BW máximo (Mbits/s)	Desv. típica	BW máximo (Mbits/s)	Desv. típica
Bare metal	94.16	0.05	22.06	0.39
Docker	94.30	0	21.66	0.39
Kubernetes	90.60	0	18.32	0.80

Por último, hay que mencionar que los *benchmarks*

`IOzone` e `iperf` no introducen una carga de CPU significativa, lo que conlleva, tanto a que el consumo de energía como las temperaturas alcanzadas por la Raspberry Pi sean mínimos.

V. Conclusiones y trabajos futuros

Es innegable que en los últimos años el concepto de Internet de las Cosas ha adquirido un gran auge, motivado por la gran cantidad de oportunidades de negocio y de investigación que ofrece al dotar de conexión a Internet a objetos cotidianos. Algunas de la aplicaciones que han surgido imponen novedosos requisitos sobre la arquitectura convencional basada en la nube, que ésta no es capaz de satisfacer. Así, ha surgido el paradigma de computación Fog, donde (al menos parte de) los recursos que han de almacenar y procesar la información de las "cosas" se encuentran cerca de ellas. Las técnicas de virtualización ligera basadas en contenedores, así como los computadores monoplaca son susceptibles de ser empleados para implementar un Fog de bajo coste y consumo energético, y prestaciones suficientes.

En este trabajo se ha presentado una primera evaluación de la sobrecarga que introducen Docker y Kubernetes sobre las prestaciones de una Raspberry Pi. Se han obtenido datos relativos al consumo de CPU, prestaciones de E/S y de red, empleando para ello diversos *benchmarks*. Los resultados presentados reflejan la idea de que Docker es ligero: apenas existe sobrecoste de cara al usuario de la aplicación ni de cara al uso de los recursos del sistema.

Por otro lado, Kubernetes sí que impone algo más de sobrecarga, pues implica la existencia de procesos en todos los nodos del despliegue, que son imprescindibles para mantener la integridad del clúster Kubernetes y los servicios que en él se desplieguen. No obstante, la sobrecarga introducida no es excesiva, si se tienen en cuenta las capacidades que ofrece.

Como trabajo futuro se pretende extender el análisis realizado implementando casos de uso reales, con especial interés en aplicaciones que soporten de técnicas de *deep learning* y *machine learning* sin necesidad de hacer uso de los recursos del cloud. Además, se está trabajando en proveer una arquitectura Fog basada en capas que gestione los recursos de los nodos SBC de forma inteligente.

Agradecimientos

El presente trabajo ha sido financiado mediante el proyecto RTI2018-098156-B-C52, financiado por el Ministerio de Ciencia, Innovación y Universidades del Gobierno de España.

Referencias

[1] E. Borgia, "The internet of things vision: Key features, applications and open issues," *Computer Communications*, vol. 54, pp. 1 – 31, 2014.

[2] "The industrial internet: Towards the 4th industrial revolution," IoT Now Magazine [online], `www.iot-now.com/2016/10/20/53811-the-industrial-internet-towards-the-4th-industrial-revolution`, 2016.

[3] Cisco Devnet, "What is IOx?," `https://developer.cisco.com/docs/iox/`, accedido en mayo, 2019.

[4] L. Dignan, "What's next for data centers? Think micro data centers," `https://www.zdnet.com/article/whats-next-for-data-centers-think-micro-data-centers/`, accedido en mayo, 2019.

[5] Raspberry Pi Foundation, "Página oficial de Raspberry Pi," `https://www.raspberrypi.org`, accedido en mayo, 2019.

[6] "Container Definition," `https://techterms.com/definition/container`, accedido en mayo, 2019.

[7] "AWS IoT," `https://aws.amazon.com/es/iot/`, accedido en mayo, 2019.

[8] "Azure IoT," `https://azure.microsoft.com/es-es/overview/iot/`, accedido en mayo, 2019.

[9] "Google Cloud IoT," `https://cloud.google.com/solutions/iot/?hl=es`, accedido en mayo, 2019.

[10] M. S. de Brito y otros, "Application of the Fog computing paradigm to Smart Factories and cyber-physical systems," *Transactions on Emerging Telecommunications Technologies*, 5 2017.

[11] L. Gu y otros, "Cost Efficient Resource Management in Fog Computing Supported Medical Cyber-Physical System," *IEEE Transactions on Emerging Topics in Computing*, , no. 1, pp. 108, 2017.

[12] F. Ramalho and A. Neto, "Virtualization at the network edge: A performance comparison," in *IEEE 17th International Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, 2016.

[13] P. Bellavista and A. Zanni, "Feasibility of Fog Computing Deployment based on Docker Containerization over RaspberryPi," in *18th International Conference on Distributed Computing and Networking*, 2017.

[14] L. Yin, J. Luo, and H. Luo, "Tasks Scheduling and Resource Allocation in Fog Computing Based on Containers for Smart Manufacturing," *IEEE Transactions on Industrial Informatics*, , no. 10, pp. 4712, 2018.

[15] Inc. Docker, "What is a Container?," `https://www.docker.com/what-container`, accedido en mayo, 2019.

[16] C. Mouradian, D. Naboulsi, S. Yangui, R. H. Glitho, M. J. Morrow, and P. A. Polakos, "A comprehensive survey on fog computing: State-of-the-art and research challenges," *IEEE Communications Surveys Tutorials*, vol. 20, no. 1, pp. 416–464, Firstquarter 2018.

[17] R. Buyya and A. V. Dastjerdi, *Internet of Things - Principles and Paradigms*, Morgan-Kauffman, 2016.

[18] R. Morabito, "A performance evaluation of container technologies on Internet of Things devices," in *IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2016.

[19] Inc. Docker, "Enterprise Container Platform for High-Velocity Innovation," `https://www.docker.com`, accedido en mayo, 2019.

[20] The Kubernetes Authors, "Kubernetes: Production-Grade Container Orchestration," `https://kubernetes.io/`, accedido en mayo, 2019.

[21] Docker Docs, "Use the OverlayFS storage driver," `https://docs.docker.com/storage/storagedriver/overlayfs-driver/`, accedido en junio, 2019.

[22] Lindong Liu, Deyu Qi, Naqin Zhou, and Yilin Wu, ""a task scheduling algorithm based on classification mining in fog computing environment"," vol. 2018, pp. 18.

[23] R. Morabito, "Virtualization on Internet of Things Edge Devices With Container Technologies: A Performance Evaluation," *IEEE Access*, vol. 5, pp. 8835–8850, 2017.

Una nueva plataforma social para el procesamiento de imágenes hiperespectrales de manera masiva

Miguel Blanco[1], Mercedes E. Paoletti[1], Juan M. Haut[1], Javier Plaza[1] y Antonio Plaza[1]

Resumen— **El uso de las redes sociales aumenta cada día. Esto, sumado a las mejoras en las tecnologías y nuevos mecanismos de obtención de información del espacio, nos permiten participar de una manera activa y aprender rápidamente sobre nuestro planeta. Cada usuario de una red social puede actuar como un sensor tanto activo como pasivo de la Tierra, trabajando tanto con información proveniente de satélites de sensores remotos, como con datos que el propio usuario aporte. Todos estos elementos nos proveen nuevas opciones para la interacción entre diferentes usuarios. La unión de las aplicaciones de mensajería con las nuevas tecnologías que aparecen dentro del mundo de la teledetección nos aporta nueva información que es necesario procesar. En este trabajo, presentamos una nueva herramienta basada en *Telegram*, para procesar imágenes hiperespectrales de la superficie terrestre. El trabajo supone una novedad, ya que no existen herramientas similares en la literatura.**

Palabras clave— **Big data, procesamiento masivo, imagen hiperespectral**

I. Introducción

El número de usuarios y tiempo de uso de las redes sociales sigue creciendo cada día. Esto, unido a la gran mejora de las tecnologías espaciales nos abre la posibilidad de nuevas funciones como la tarea de localizar en cualquier momento y lugar a diferentes usuarios, y la posibilidad de participar e interactuar con ellos, de una forma nunca antes conocida. Los usuarios de estas tecnologías pueden aportar nuevos datos sobre el estado del entorno físico del planeta.

Esta información, que puede provenir de satélites de sensores remotos, sistemas de información geográfica (SIG) o de los propios usuarios, abre nuevas posibilidades para poder comprender cómo un grupo de personas pueden organizarse para conseguir un impacto social, y además, sientan las bases para los nuevos sistemas sociales de uso popular. La confluencia de las tecnologías de teledetección [1] con el añadido del aumento del uso de las redes sociales, nos está llevando hacia una situación en la que podemos saber de una manera sencilla y eficaz dónde se ubican todas las personas en cualquier momento, y explotar el poder del *crowdsourcing* [2] de las redes sociales en diferentes contextos. Todo este conjunto de información, también llamada *big data* [3], precisa de técnicas avanzadas de aprendizaje automático [4], además de una correcta infraestructura informática de alto rendimiento.

[1]Hyperspectral Computing Laboratory, Department of Technology of Computers and Communications, Escuela Politécnica, University of Extremadura, 10003 Cáceres, Spain (e-mail: mpaoletti@unex.es; juanmariohaut@unex.es; jplaza@unex.es; aplaza@unex.es)

El aumento del uso y el creciente impacto de las redes sociales necesita de la creación de nuevos sistemas de información social. Para conseguir esto, es necesario procesar grandes cantidades de datos, formados por información proveniente de las redes sociales e imágenes de satélite, por lo que se exigirá la creación de nuevos marcos de procesamiento de datos además de nuevos algoritmos, nuevos métodos y nuevos sistemas. Uno de los principales objetivos al integrar todos estos nuevos tipos de procesamiento de datos es el de reducir el tiempo en obtener los resultados [5][6]. Para conseguir esta ganancia de velocidad, es necesario el uso de procesadores cada vez más rápidos. Debido al creciente interés en las redes sociales y la tecnología espacial, es necesario explorar nuevas perspectivas para la unión de estas tecnologías, utilizando técnicas avanzadas de aprendizaje automático.

Las imágenes remotas se recogen a través de sensores situados en plataformas aerotransportadas [7]. Estos sensores pueden ser activos o pasivos. Un sensor activo es un sensor que genera artificialmente su propia radiación y la mide a continuación [8]. Los sensores pasivos son aquellos que reciben la radiación electromagnética emitida o reflejada por la Tierra [8]. El problema de estos sensores es que no están siempre disponibles cuando se producen sucesos como pueden ser terremotos o incendios, lo que produce una ralentización a la hora de tomar decisiones. Debido a las restricciones de hardware, convertir a los usuarios que se encuentren en la zona afectada en sensores tanto activos como pasivos tiene el potencial de complementar la carencia temporal de las plataformas aerotransportadas. Se han logrado numerosos avances en los últimos años en cuanto a la recolección, procesamiento y almacenamiento tanto de datos espaciales como de datos provenientes de las redes sociales [9]. No obstante, la explotación de toda esta información para una correcta y veloz toma de decisiones a la hora de reaccionar ante un suceso requiere nuevos desarrollos computacionales.

En este trabajo, proporcionamos una nueva visión de los avances más recientes en las áreas de investigación antes mencionadas. Aportando nuestro particular enfoque sobre la combinación de datos que provienen de sensores remotos y de información de las redes sociales, presentamos un nuevo enfoque para la combinación de redes sociales y tecnologías espaciales, detección y localización, monitoreo y *big data*, que nunca antes se han abordado de forma conjunta.

En concreto, presentamos una una nueva herra-

mienta computacionalmente eficiente, basada en *Telegram*, para procesar imágenes hiperespectrales de la superficie terrestre. El resto del artículo se organiza de la siguiente manera. En la Sección II se presenta el diseño de la arquitectura del sistema que hemos implementado. Se encuentra estructurado en tres partes: 1) cliente, 2) servidor de *Telegram* y 3) servidor de la aplicación, proporcionándose una explicación detallada de cada una de ellas, su contenido y comunicaciones que realiza. La sección III explica las diferentes operaciones disponibles en la aplicación de una manera breve y concisa, además de los algoritmos con los que se trabaja y como están estructurados. En la sección IV, se presentan los resultados experimentales, obtenidos con la conocida imagen hiperespectral *Indian Pines* [10]. Por último, en la sección V se comentan las principales conclusiones y se indican futuras líneas de trabajo, así como una previsión de futuro de nuestra aplicación.

II. Metodología

Esta sección describe de manera detallada el sistema que hemos decidido implementar, el cuál, se encuentra estructurado en tres partes tal y como se muestra en la Fig. 1.

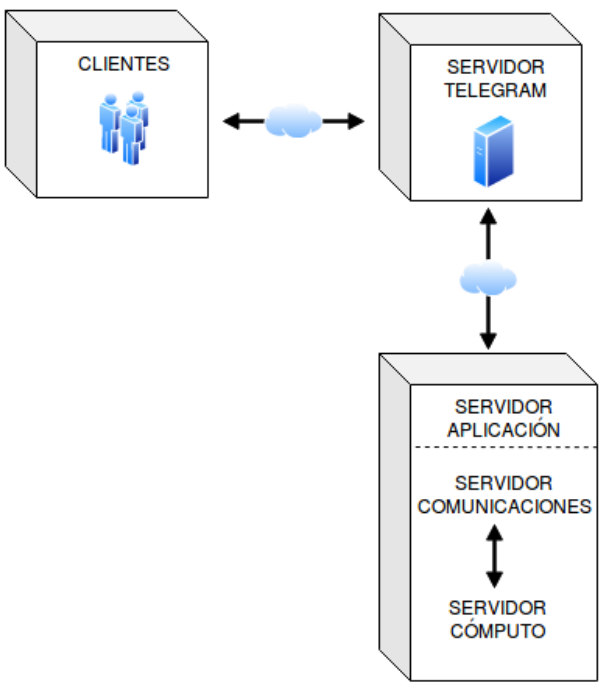


Fig. 1. Diseño de la arquitectura del sistema.

Como se puede apreciar en la Fig. 1, la arquitectura propuesta se basa en capas completamente independientes entre ellas desde un punto de vista de alto nivel. El sistema es *plug-and-play*, lo que permite la incorporación de nuevos algoritmos y nuevas funcionalidades [11]. Gracias a este diseño, cualquier parte puede ser reemplazada y además, esto se realiza de forma transparente al usuario. El diseño de la parte encargada del procesamiento de los diferentes algoritmos también utiliza un diseño de *plug-and-play* [11], lo que nos permite la incorporación de nuevos componentes de manera sencilla sin tener que modificar la aplicación. La comunicación entre los diferentes bloques se realiza a través de Internet, de una forma cifrada a través del protocolo *MTProto* [12], el cual permite el intercambio entre dos dispositivos de una clave utilizando el intercambio de claves Diffie-Hellman [13]. Como resultado, todas las partes se encuentran disponibles desde cualquier ubicación, y el rendimiento depende del tiempo de procesamiento del algoritmo elegido. A continuación, pasamos a describir cada parte en detalle.

A. Cliente

En esta zona se encuentra alojada la instalación de Telegram que será la herramienta con la que interactúe el usuario. Se trata de una aplicación multiplataforma, por lo que su adquisición y posterior instalación puede realizarse a través de sus repositorios oficiales o la plataforma de distribución digital que tenga el dispositivo. El cliente inicia una comunicación con Telegram, mandando una petición de solicitud de comunicación entre el cliente y nuestra aplicación. El cliente recibe el menú con las diferentes operaciones y algoritmos, y sólo debe seleccionar y enviar sus elecciones. Todas las acciones del cliente se ejecutan en el servidor de la aplicación, por lo que servidor de Telegram sólo actúa de intermediario entre estos dos. El cliente selecciona la operación que desea ejecutar, y recibe los algoritmos que pertenecen a esa operación. Después la aplicación le preguntará por el dataset que desea procesar, y puede seleccionarlo desde nuestra biblioteca pública, nuestra biblioteca privada, buscar una imagen en Google, o incluso aportar el suyo propio. La aplicación de Telegram incorpora una funcionalidad para buscar imágenes en los motores de búsqueda de Bing o Yandex, que también puede ser usada. Posteriormente, se le requerirá algún aporte extra informando sobre los parámetros predeterminados, y deberá esperar a recibir la imagen resultante del proceso de ejecución del algoritmo con el dataset escogido.

B. Servidor de Telegram

El servidor de Telegram se encarga de mantener las comunicaciones y realiza las labores de intermediario entre el cliente y el servidor de la aplicación. Se encarga también de cifrar los mensajes y de que éstos no se vean comprometidos por un agente externo. Implementa métodos para comprobar que el mensaje no se ha visto alterado, y gestiona la conexión de forma transparente al usuario. Para identificar a uno u otro de forma anónima, envía un identificador único a cada uno de ellos por lo que su comunicación se realiza a través de estos identificadores. Además, el servidor almacena todas las consultas y mensajes que pasan a través de él en su base de datos privada. Si el cliente envía un dataset, debe pasar por la parte de Telegram primero, aunque no tiene restricciones (en cuanto a tamaño o extensión) a la hora de enviar archivos.

C. Servidor de la aplicación

Esta es la parte más importante del sistema. Está formado, a su vez, por dos partes diferentes. La primera se encarga de las comunicaciones entre esta parte y *Telegram*, y la segunda es un servidor de cómputo encargado de la ejecución del algoritmo seleccionado por el usuario, utilizando el *dataset* y los parámetros recibidos. A continuación describimos estas dos partes.

1. El servidor de comunicaciones es el encargado de manejar y procesar el tráfico entrante que proviene de *Telegram* y además aloja las imágenes y *datasets* de nuestra biblioteca pública y privada. Permite generar un menú con las siguientes técnicas de extracción de información: clasificación [14] [15], reducción de la dimensionalidad [16], superresolución [17] y *unmixing* [18] [19]. Recibe las opciones que el cliente ha ido seleccionando. Si el cliente ha decidido trabajar con una imagen o *dataset* aportado por él, recibirá esta información proveniente de *Telegram* y la almacenará en nuestra biblioteca privada, imponiendo como nombre del archivo la fecha y hora actual. Una vez que ha terminado de recibir todos los parámetros, crea el mensaje que enviará al servidor de cómputo para su procesamiento, que estará formado por el algoritmo y *dataset* escogidos, además de los parámetros necesarios y el identificador del usuario. Esperará a que el servidor de cómputo termine de procesar los datos y enviará a *Telegram* la imagen resultante y los datos que sean necesarios para que éste se los devuelva al cliente. Esta parte fue desarrollada bajo el sistema operativo *Ubuntu* con el lenguaje de programación *Python*.
2. El servidor de cómputo se encarga de administrar y procesar las diferentes operaciones que el usuario puede seleccionar, así como de hospedar los algoritmos implementados. Está diseñado de manera que las operaciones y algoritmos se tratan como componentes. Recibe el dataset y los parámetros que el usuario ha seleccionado y ejecuta el algoritmo de procesamiento que corresponda. Una vez finalizada la ejecución del algoritmo, almacena la consulta del cliente y su resultado en nuestra base de datos, y devuelve la dirección de la información procesada (así como el identificador del cliente que realizó la consulta) al servidor de comunicaciones. Esta otra parte ha sido desarrollada utilizando el lenguaje de programación Python haciendo uso de la librería *sklearn* [20], una biblioteca de aprendizaje automático de software (gratuita) que cuenta con varios de los algoritmos que hemos implementado y está diseñada para trabajar con las librerías numéricas y científicas *NumPy* [21] y *SciPy* [22]. Para implementar algunas operaciones adicionales a a la hora de tratar las imágenes, se optó por el uso en su desarrollo de *OpenCV* [23], una biblioteca libre de visión artificial para imágenes.

Estas dos partes se encuentran alojadas en la misma máquina física para reducir el tiempo de comunicación entre ellas, reduciendo considerablemente el tiempo de ejecución total desde que el cliente inicia una comunicación hasta que recibe el resultado final.

III. Algoritmos implementados

Los algoritmos que hemos introducido en nuestra aplicación se pueden englobar en cuatro grandes grupos:

- **Clasificación** [14] [15]: La operación de clasificación se refiere a la agrupación de los píxeles de una imagen en diferentes clases [24]]. Los algoritmos que hemos implementado dentro de esta categoría son *K-means* [25] y *K-nearest neighbors KNN* [26].
- **Reducción de dimensionalidad** [16]: La operación de reducción de dimensionalidad se refiere al proceso de disminuir la dimensionalidad de la imagen original mediante la extracción de componentes principales. Los algoritmos introducidos de esta operación son *Principal Component Analysis* (PCA) [27] [28] y *Maximum Noise Fraction* (MNF) [29].
- **Superresolución** [17]: Esta operación se refiere al proceso de redimensionar una imagen utilizando una escala que el usuario especifica. Se utilizan tres tipos de interpolación: bicúbica [30], *Nearest Neighbor* [30] y Lanczos [31].
- **Unmixing** [18] [19]: Proceso de descomponer cada imagen en un conjunto de firmas espectrales puras o *endmembers* y sus correspondientes mapas de abundancia. Incluye los algoritmos *Fully Constrained Linear Spectral Unmixing* (FCLSU) [32], *Linear Discriminant Analsys* (LDA) [33] y *Vertex Component Analysis* (VCA) [34].

IV. Resultados

Las imágenes que hemos utilizado durante los experimentos, han sido adquiridas mediante sensores hiperespectrales. Además, hemos utilizado también imágenes obtenidas a partir de *Google Earth*. En este apartado ilustramos la ejecución de varios algoritmos sobre el mismo dataset, haciendo uso de la imagen *Indian Pines* [10] obtenida por el sensor hiperespectral *Airborne Visible Infra-Red Imaging Spectrometer* (AVIRIS) [35], que cubre una zona agrícola del noroeste del estado de Indiana, Estados Unidos. La imagen consta de 200 bandas espectrales. Dos tercios de la imagen incluyen zonas agrícolas, y el resto es bosque y vegetación.

En cuanto a la interacción del usuario con nuestra aplicación, hemos desarrollado una interfaz gráfica intuitiva y sencilla (ver Fig. 2) en la que el usuario debe escoger la operación y algoritmo deseado, permitiendo la posibilidad de interactuar con otros usuarios, utilizando su agenda de contactos para enviar y comentar los resultados obtenidos.

Para ilustrar el resultado de la ejecución de uno de algoritmos, estudiando el tiempo que tarda la apli-

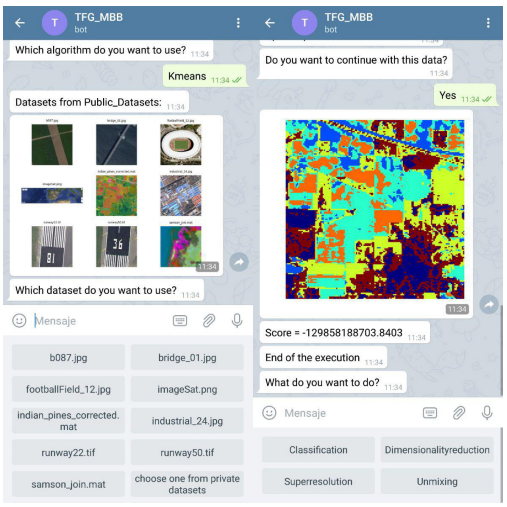


Fig. 2. Interfaz del bot de Telegram y ejemplo de ejecución utilizando la imagen hiperespectral AVIRIS Indian Pines.

cación desde que se le manda la petición hasta que recibe el resultado del procesamiento, mostramos el resultado de la ejecución de KNN [26] con K=16 vecinos (ver Fig. 3. El tiempo que tarda el algoritmo en procesar la imagen hiperespectral AVIRIS Indian Pines es 0.0067 segundos.

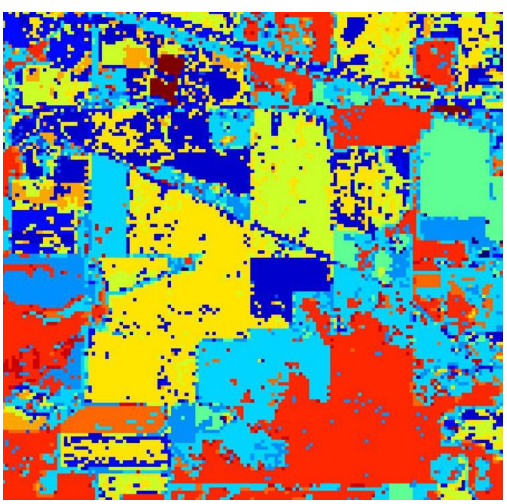


Fig. 3. Resultado de aplicar el algoritmo KNN sobre la imagen AVIRIS Indian Pines con K=16 vecinos.

También mostramos el resultado de aplicar el algoritmo PCA [27] [28] con N=20 componentes sobre esta misma imagen en la Fig. 4. En este caso, el tiempo de ejecución del algoritmo es de 0.0284 segundos.

Finalmente, mostramos un ejemplo de ejecución de *K-means* [25] sobre una imagen tomada con el móvil por el usuario, escogiendo un número de 5 clústers: Para ello el usuario debe seleccionar la opción de *upload my own dataset* cuando se le pregunta por el *dataset* a procesar y abre su cámara a través de la propia aplicación de *Telegram*, enviando la fotografía en ese mismo momento para su procesamiento, como se puede ver en la Fig. 5.

V. Conclusiones y trabajo futuro

En este trabajo hemos descrito hemos descrito una nueva herramienta basada en *Telegram* para el procesamiento de imágenes de satélite, la cual integra las redes sociales con tecnologías de detección de objetos remotos. La aplicación permite aplicar varios algoritmos de extracción de información sobre imágenes disponibles y también sobre imágenes capturadas por el móvil. En el futuro, planeamos continuar mejorando este sistema, añadiendo nuevas funcionalidades y características. Por ejemplo, un posible nuevo avance sería incluir los resultados a *Twitter* o a través de un canal de *Telegram*, lo que permitiría ver si la consulta que se quiere realizar ha sido antes resuelta, o incluso, dar la posibilidad al usuario de que su consulta no sea guardada en nuestra base de datos para obtener una mayor privacidad. Otra mejora interesante sería la de incluir nuevos algoritmos. También se podrían incluir nuevos motores de búsqueda (actualmente se encuentran implementados los de *Google*, *Yandex* y *Bing*). Además, la utilización de hardware de alto rendimiento podría ser interesante para procesar imágenes con muchas bandas espectrales.

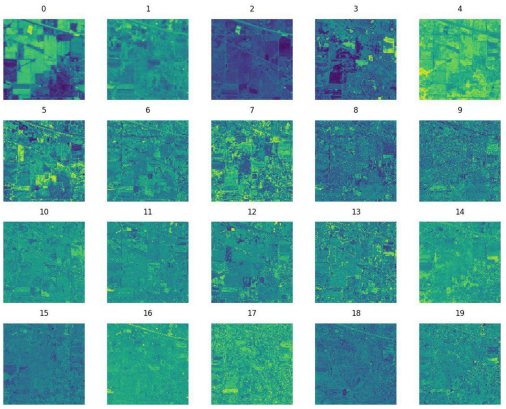


Fig. 4. Resultado de aplicar el algoritmo PCA sobre la imagen AVIRIS Indian Pines para extraer componentes principales.

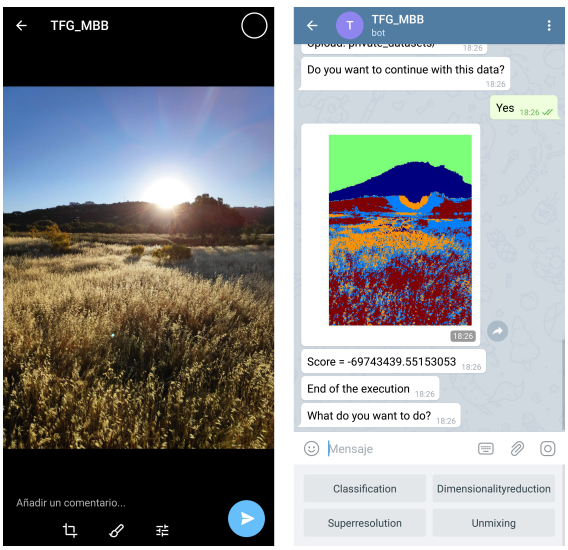


Fig. 5. Ejemplo de captación y procesamiento de una imagen tomada en ese mismo momento por el móvil.

Agradecimientos

Este trabajo ha sido financiado por el Ministerio de Educación (Resolución de 26 de diciembre de 2014

y de 19 de noviembre de 2015, de la Secretaría de Estado de Educación, Formación Profesional y Universidades, por la que se convocan ayudas para la formación de profesorado universitario, de los subprogramas de Formación y de Movilidad incluidos en el Programa Estatal de Promoción del Talento y su Empleabilidad, en el marco del Plan Estatal de Investigación Científica y Técnica y de Innovación 2013-2016). Este trabajo también ha sido financiado por la Junta de Extremadura (Decreto 14/2018, de 6 de febrero, por el que se establecen las bases reguladoras de las ayudas para la realización de actividades de investigación y desarrollo tecnológico, de divulgación y de transferencia de conocimiento por los Grupos de Investigación de Extremadura, Ref. GR18060). El trabajo también ha sido parcialmente financiado por el programa Horizonte 2020 de la Unión Europea, Research and Innovation Programme, Grant No. 734541 (EOXPOSURE).

Referencias

[1] James B Campbell and Randolph H Wynne, *Introduction to remote sensing*, Guilford Press, 2011.

[2] Enrique Estellés-Arolas and Fernando González-Ladrón-De-Guevara, "Towards an integrated crowdsourcing definition," *Journal of Information science*, vol. 38, no. 2, pp. 189–200, 2012.

[3] Amir Gandomi and Murtaza Haider, "Beyond the hype: Big data concepts, methods, and analytics," *International journal of information management*, vol. 35, no. 2, pp. 137–144, 2015.

[4] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al., "Scikit-learn: Machine learning in python," *Journal of machine learning research*, vol. 12, no. Oct, pp. 2825–2830, 2011.

[5] Antonio Plaza, Qian Du, Yang-Lang Chang, and Roger L King, "High performance computing for hyperspectral remote sensing," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 4, no. 3, pp. 528–544, 2011.

[6] Craig A Lee, Samuel D Gasster, Antonio Plaza, Chein-I Chang, and Bormin Huang, "Recent developments in high performance computing for remote sensing: A review," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 4, no. 3, pp. 508–527, 2011.

[7] Wenzhi Zhao and Shihong Du, "Spectral–spatial feature extraction for hyperspectral image classification: A dimension reduction and deep learning approach," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 54, no. 8, pp. 4544–4554, 2016.

[8] Klaus Erdle, Bodo Mistele, and Urs Schmidhalter, "Comparison of active and passive spectral sensors in discriminating biomass parameters and nitrogen status in wheat cultivars," *Field Crops Research*, vol. 124, no. 1, pp. 74 – 84, 2011.

[9] Amir Hussain and Erik Cambria, "Semi-supervised learning for big social data analysis," *Neurocomputing*, vol. 275, pp. 1662–1673, 2018.

[10] NW Aviris, "Indiana's indian pines 1992 data set," 2012.

[11] Wesley Chun, *Core python programming*, vol. 1, Prentice Hall Professional, 2001.

[12] Jakob Jakobsen and Claudio Orlandi, "On the cca (in) security of mtproto.," in *SPSM@ CCS*, 2016, pp. 113–116.

[13] Michael Steiner, Gene Tsudik, and Michael Waidner, "Diffie-hellman key distribution extended to group communication," in *Proceedings of the 3rd ACM Conference on Computer and Communications Security*, New York, NY, USA, 1996, CCS '96, pp. 31–37, ACM.

[14] Chein-I Chang, *Hyperspectral imaging: techniques for spectral detection and classification*, vol. 1, Springer Science & Business Media, 2003.

[15] Chein-I Chang, *Hyperspectral data exploitation: theory and applications*, John Wiley & Sons, 2007.

[16] Joseph C Harsanyi and C-I Chang, "Hyperspectral image classification and dimensionality reduction: An orthogonal subspace projection approach," *IEEE Transactions on geoscience and remote sensing*, vol. 32, no. 4, pp. 779–785, 1994.

[17] Toygar Akgun, Yucel Altunbasak, and Russell M Mersereau, "Super-resolution reconstruction of hyperspectral images," *IEEE Transactions on Image Processing*, vol. 14, no. 11, pp. 1860–1875, 2005.

[18] José M Bioucas-Dias, Antonio Plaza, Nicolas Dobigeon, Mario Parente, Qian Du, Paul Gader, and Jocelyn Chanussot, "Hyperspectral unmixing overview: Geometrical, statistical, and sparse regression-based approaches," *IEEE journal of selected topics in applied earth observations and remote sensing*, vol. 5, no. 2, pp. 354–379, 2012.

[19] Lucas C Parra, Clay Spence, Paul Sajda, Andreas Ziehe, and Klaus-Robert Müller, "Unmixing hyperspectral data," in *Advances in neural information processing systems*, 2000, pp. 942–948.

[20] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al., "Scikit-learn: Machine learning in python," *Journal of machine learning research*, vol. 12, no. Oct, pp. 2825–2830, 2011.

[21] Stefan Van Der Walt, S Chris Colbert, and Gael Varoquaux, "The numpy array: a structure for efficient numerical computation," *Computing in Science & Engineering*, vol. 13, no. 2, pp. 22, 2011.

[22] Eric Jones, Travis Oliphant, and Pearu Peterson, "{SciPy}: Open source scientific tools for {Python}," 2014.

[23] Gary Bradski and Adrian Kaehler, *Learning OpenCV: Computer vision with the OpenCV library*, " O'Reilly Media, Inc.", 2008.

[24] Boris Mirkin, *Mathematical classification and clustering*, vol. 11, Springer Science & Business Media, 2013.

[25] John A Hartigan and Manchek A Wong, "Algorithm as 136: A k-means clustering algorithm," *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, vol. 28, no. 1, pp. 100–108, 1979.

[26] Thomas M Cover, Peter E Hart, et al., "Nearest neighbor pattern classification," *IEEE transactions on information theory*, vol. 13, no. 1, pp. 21–27, 1967.

[27] Ian Jolliffe, *Principal component analysis*, Springer, 2011.

[28] Michael E Wall, Andreas Rechtsteiner, and Luis M Rocha, "Singular value decomposition and principal component analysis," in *A practical approach to microarray data analysis*, pp. 91–109. Springer, 2003.

[29] Daniel D Lee and H Sebastian Seung, "Algorithms for non-negative matrix factorization," in *Advances in neural information processing systems*, 2001, pp. 556–562.

[30] Kenneth Dawson-Howe, *A practical introduction to computer vision with opencv*, John Wiley & Sons, 2014.

[31] George J Grevera and Jayaram K Udupa, "Shape-based interpolation of multidimensional grey-level images," *IEEE transactions on medical imaging*, vol. 15, no. 6, pp. 881–892, 1996.

[32] Daniel C Heinz et al., "Fully constrained least squares linear spectral mixture analysis method for material quantification in hyperspectral imagery," *IEEE transactions on geoscience and remote sensing*, vol. 39, no. 3, pp. 529–545, 2001.

[33] Qian Du and Chein-I Chang, "A linear constrained distance-based discriminant analysis for hyperspectral image classification," *Pattern Recognition*, vol. 34, no. 2, pp. 361–373, 2001.

[34] José MP Nascimento and José MB Dias, "Vertex component analysis: A fast algorithm to unmix hyperspectral data," *IEEE transactions on Geoscience and Remote Sensing*, vol. 43, no. 4, pp. 898–910, 2005.

[35] Robert O Green, Michael L Eastwood, Charles M Sarture, Thomas G Chrien, Mikael Aronsson, Bruce J Chippendale, Jessica A Faust, Betina E Pavri, Christopher J Chovit, Manuel Solis, et al., "Imaging spectroscopy and the airborne visible/infrared imaging spectrometer (aviris)," *Remote sensing of environment*, vol. 65, no. 3, pp. 227–248, 1998.

[36] Jakob Bjerre Jakobsen and Claudio Orlandi, *A prac-*

tical cryptanalysis of the Telegram messaging protocol, Ph.D. thesis, PhD thesis, Master Thesis, Aarhus University (Available on request), 2015.

Implementación de metaheurísticas paralelas en entornos cloud.

Diego Teijeiro, Patricia González, Xoán C. Pardo y Ramón Doallo[1]

Resumen— **En muchas áreas de ciencia e ingeniería las metaheurísticas se encuentran entre los métodos más populares para resolver problemas difíciles de optimización global. Su implementación paralela, usando técnicas de computación de altas prestaciones (HPC) es habitual, con el objetivo de usar de forma eficiente los recursos disponibles reduciendo el tiempo necesario para alcanzar soluciones suficientemente buenas a problemas de elevada dificultad. Paradigmas como MPI u OMP son las elecciones habituales cuando se ejecutan en clusters o superordenadores. Además, la presencia cada vez más amplia de la computación cloud, y la aparición de modelos de programación como MapReduce o Spark, han generado un interés creciente por migrar aplicaciones HPC a la nube, como en el caso de las metaheurísticas paralelas. En este trabajo proporcionamos una visión general de nuestra experiencia con distintas alternativas para portar metaheurísticas paralelas, como el método de evolución diferencial (DE), al cloud, junto a las conclusiones y lecciones aprendidas de nuestros experimentos.**

Palabras clave—**Computación en la nube, metaheurísticas paralelas, MPI, MapReduce, Spark**

I. Introducción

Los métodos de optimización se centran en encontrar la *mejor solución disponible* para un problema dado. Muchos desafíos claves en diferentes campos de ciencia, economía e ingeniería se pueden formular y resolver usando diferentes técnicas de optimización. Por ejemplo, los problemas de optimización están jugando un rol cada vez más importante en biología computacional, bioinformática y química, ayudando en el desarrollo de nuevas terapias y medicamentos para distintas enfermedades como cáncer o enfermedades auto-inmunes. La mayor parte de estos problemas de optimización son, en la práctica, complejos, con dificultad NP y costosos en tiempo. Los métodos estocásticos de optimización global son una alternativa robusta para resolver estos problemas. Entre ellos, las metaheurísticas están cobrando cada vez mayor relevancia como una manera de resolver de forma eficiente problemas de optimización global. Sin embargo, en la mayoría de aplicaciones realistas, las metaheurísticas siguen necesitando largos tiempos de ejecución para alcanzar resultados aceptables.

Gracias a los avances tecnológicos de la última década, el uso de la computación paralela ha ganado popularidad de forma progresiva. Además, el ratio coste/rendimiento de las arquitecturas HPC se sigue reduciendo, más rápido incluso con la aparición de infraestructuras cloud, con lo que obtener acceso a recursos de altas prestaciones se vuelve cada vez más sencillo. Por tanto, no sorprende que la paralelización de los métodos de optimización en general, y de metaheurísticas en particular, haya recibido cada vez más atención como una forma de reducir el tiempo de solución en problemas de gran escala. En la literatura se han propuesto y descrito multitud de algoritmos paralelos y se pueden encontrar buenos análisis en [1], [2]. Sin embargo, la mayor parte de estas propuestas se basan en interfaces y paradigmas tradicionales, como MPI u OpenMP, ejecutándose en infraestructuras tradicionales como clusters o superordenadores. Alcanzar una paralelización eficiente de estos algoritmos usando estos paradigmas no es una tarea sencilla, dado que la búsqueda de nuevas soluciones depende de iteraciones anteriores, lo que no solo complica la paralelización si no que también limita la escalabilidad.

El resto del trabajo se estructura de la siguiente manera. La sección II resume brevemente el trabajo relacionado en el campo. La sección III detalla las distintas alternativas que se han probado para la paralelización del algoritmo. En la sección IV se muestran los resultados obtenidos con las distintas implementaciones probadas. En la sección V se exponen las conclusiones alcanzadas después de la experimentación realizada.

II. Trabajo relacionado

En la última década, algunos investigadores han prestado especial atención y evaluado el rendimiento de aplicaciones paralelas en el cloud [3], [4], llegando a la conclusión de que las plataformas cloud no se han diseñado para ejecutar aplicaciones paralelas fuertemente acopladas. El menor ancho de banda, la latencia de la red de interconexión, así como el sobrecoste de la virtualización, tienen un gran impacto en el rendimiento de este tipo de aplicaciones en el cloud. Algunos (pocos) trabajos también has estudiado la paralelización de metaheurísticas para ser ejecutadas en entornos Cloud [5], [6]. La mayor parte de estos trabajos están basados principalmente en el uso de MapReduce [7], dado que era el estándar de facto. Sin embargo, los resultados de estos estudios revelan que el coste adicional de las operaciones de entrada/salida y el sobrecoste del sistema reducen significativamente el rendimiento de la paralelización. Esto se debe a que MapReduce no es un paradigma adecuado para algoritmos iterativos, como es el caso de las metaheurísticas. El rendimiento puede mejorarse empleando otros entornos de computación en memoria con mejor soporte para algoritmos iterativos, como Spark [8] o Flink [9].

[1]Grupo de Arquitectura de Computadores. CITIC. Universidade da Coruña. Spain, e-mail: patricia.gonzalez@udc.es

III. Implementación paralela del método DE

En trabajos previos [10], [11], [12], hemos explorado el uso de MPI, orientado a computación de altas prestaciones, y Hadoop y Spark que están orientados a rendimiento, en la implementación de metaheurísticas basadas en poblaciones.

Para nuestro estudio hemos seleccionado una metaheurística representativa, basada en población, que se usará como caso de prueba, la Evolución Diferencial (Differential Evolution - DE) [13]. El método DE es muy popular y se ha aplicado con éxito a muchos problemas [14]. Es un método iterativo estocástico (ver algoritmo 1) que partiendo de una matriz poblacional inicial formada por NP vectores de soluciones D-Dimensionales (individuos), trata de alcanzar la solución óptima de forma iterativa empleando diferencias de vectores para crear nuevas soluciones candidatas. En cada iteración, se generan nuevos individuos mediante operaciones (cruce - CR; mutación - F) realizadas entre entradas en la matriz de población. Las nuevas soluciones reemplazarán a las antiguas cuando el valor de fitness sea mejor. Una matriz de población con los individuos optimizados forma la salida del algoritmo. El mejor individuo se selecciona como aquel que está más cerca del óptimo de acuerdo a la función objetivo del modelo. Se han propuesto múltiples modelos para la paralelización de metaheurísticas [2]. Nosotros hemos considerado los más populares:

1. Modelo *maestro-esclavo*, que mantiene el comportamiento del algoritmo secuencial paralelizando el bucle interno del DE. En este modelo un proceso maestro distribuye las operaciones computacionales entre los procesos esclavos.
2. Modelo *basado en islas*, en el que se divide la población en subpoblaciones (islas) en las cuales se ejecuta el algoritmo de DE de forma aislada. Para mantener la diversidad y evitar quedarse atascado en mínimos locales se realizan intercambios esporádicos de elementos entre las islas.

Nuestra primera aproximación fue paralelizar DE empleando el modelo maestro-esclavo. Durante la implementación empleando Spark, nos encontramos con un problema en la paralelización de la generación de nuevos individuos. En la implementación de este modelo con Spark, la población está particionada y distribuida entre los nodos de trabajo. Para la mutación de cada individuo, es necesario seleccionar tres individuos aleatorios diferentes de la población completa pero, a diferencia de MPI, Spark no permite la comunicación entre nodos de trabajo. Cada nodo de trabajo sólo tiene acceso a los individuos en su partición y el nodo principal es el único que tiene acceso a la población completa. Entre las alternativas que se han probado para hacer frente a este problema, transmitir una copia de sólo lectura de toda la población a todos los nodos de trabajo en cada iteración fue la opción que ha mostrado mejores resultados en las pruebas. Incluso así, el sobrecoste de las comunicaciones introducido por esta solución no era asumible. Por ello, concluimos que el modelo maestro-esclavo no se adecúa correctamente al modelo de programación distribuido de Spark y decidimos descartarlo en favor del modelo basado en islas. Una revisión completa de la implementación del modelo maestro-esclavo y otras alternativas que se han probado para gestionar este problema se puede encontrar en [15].

Algorithm 1 Algoritmo de Evolución Diferencial

Entrada: Matriz de población P de tamaño D x NP
Salida: Matriz P con individuos optimizados

repeat
  for cada elemento i de la matriz P **do**
    Elegir aleatoriamente $r_1, r_2, r_3 \in [1, NP]$ distintos
    Elegir aleatoriamente un entero $j_r \in [1, D]$
    for $j \leftarrow 1, D$ **do**
      Elegir aleatoriamente un real $r \in [0, 1]$
      if $r \leq CR$ or $j = j_r$ **then**
        $u_i^{G+1}(j) \leftarrow x_{r1}^G(j) + F \cdot (x_{r2}^G(j) - x_{r3}^G(j))$
      else
        $u_i^{G+1}(j) \leftarrow x_i^G(j)$
      end if
    end for
    evaluar (u_i^{G+1})
    if $fitness(u_i^{G+1}) < fitness(x_i^G)$ **then**
      $x_i^{G+1} \leftarrow u_i^{G+1}$
    else
      $x_i^{G+1} \leftarrow x_i^G$
    end if
  end for
until Condiciones de parada

Las implementaciones evaluadas, por tanto, están basadas en modelos de islas, lo que permite reducir las comunicaciones entre los distintos procesos, permitiendo aplicaciones paralelas menos acopladas y, por tanto, más adecuadas a entornos en la nube. La Figura 1 muestra el esquema propuesto para la implementación empleando Spark, mientras que la Figura 2 muestra, para poder comparar, el mismo esquema para una implementación MPI. Existen diferencias claves entre las implementaciones de MPI y Spark que se muestran en estas figuras, que surgen de las características inherentes a los modelos de programación empleados en cada implementación, y más específicamente por el hecho de que Spark no permite la comunicación entre nodos de trabajo. La estrategia de migración entre islas, las etapas de sincronización y la comprobación de los criterios de parada se realizan de formas distintas en ambas implementaciones. En concreto, las partes del algoritmo que se implementan de forma diferente son:

- *La estrategia de migración.* Mientras que con MPI se emplea una estrategia con reemplazo (p.e. los mejores individuos de una isla reemplazan los peores en la isla vecina), en Spark, dado que no es posible la comunicación entre nodos de trabajo, la migración es una redistribución aleatoria de la población sin reemplazo.
- *La sincronización evolución-migración.* Con MPI los individuos se migran empleando comunicaciones asíncronas entre los procesos, de mo-

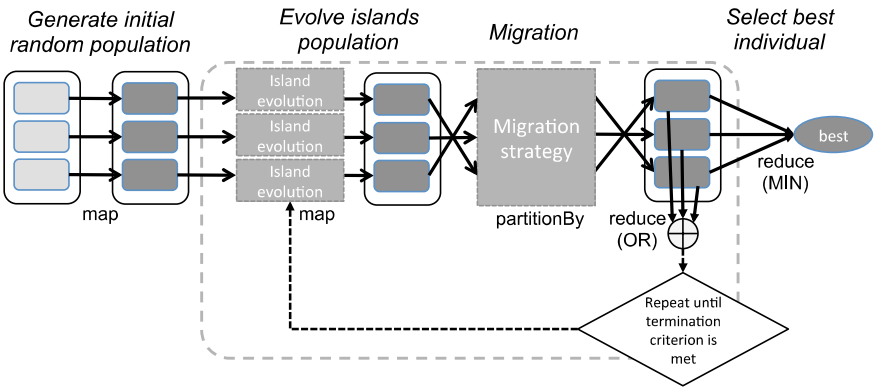


Fig. 1. Implementación con islas en Spark del método DE.

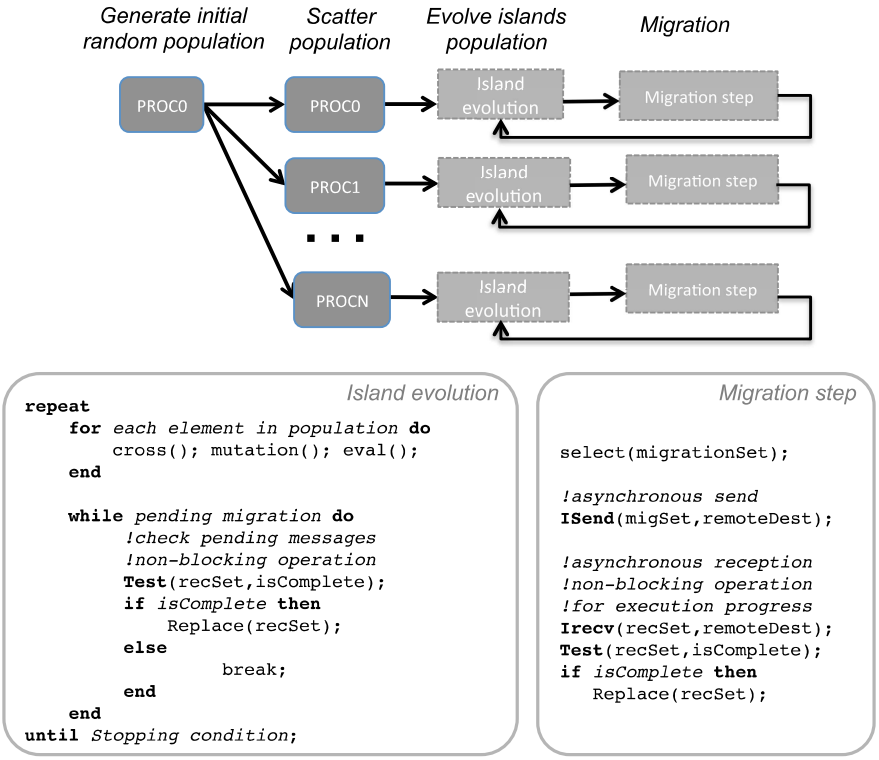


Fig. 2. Implementación con islas asíncrona en MPI del método DE.

do que cada isla puede continuar con una nueva evolución sin esperar por las demás. Por el mismo motivo que el punto anterior, en Spark esto no es posible, por lo que la migración introduce una etapa de sincronización entre las evoluciones de las islas.

- *La comprobación del criterio de parada.* Con MPI cada vez que un proceso alcanza un criterio de parada envía un mensaje asíncrono al resto de islas y todos se detienen al mismo tiempo. Una vez más, esto no es posible en Spark, de modo que tienen que esperar a que todas las islas terminen sus evoluciones para comprobar si alguna de ellas ha alcanzado la condición de parada.

Estas diferencias llevan a distintos tipos de búsquedas que modifican las propiedades sistémicas del algoritmo y, por tanto, a resultados distintos, tanto en la precisión como en el rendimiento de la metaheurística.

IV. Resultados experimentales

La evaluación del rendimiento de las diferentes implementaciones se ha realizado tanto en clusters locales como en plataformas cloud públicas como Amazon AWS y Microsoft Azure, empleando como caso de estudio un problema de estimación de parámetros en biología computacional de sistemas, el modelo Circadiano [16]. Este es un problema de estimación de parámetros en un modelo dinámico no lineal del reloj circadiano en la planta *Arabidopsis Thaliana*, conocido por ser particularmente difícil debido a su mal condicionamiento y no convexidad.

Las implementaciones analizadas han sido tres: MPI (`asynPDE`), Spark (`SiPDE`) y MapReduce (`mrPDE`). Las plataformas utilizadas han sido 4: la

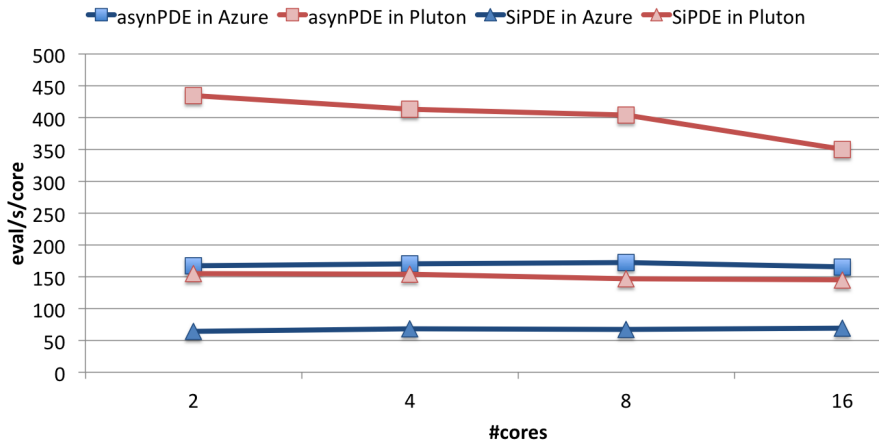


Fig. 3. Evals/s/core de las implementaciones MPI y Spark

primera un cluster local (llamado Plutón) con 16 nodos con dos procesadores Intel Xeon E5-2660 de 8 núcleos @ 2.20GHz con 64GB de memoria y una red InfiniBand FDR; la segunda un cluster virtual formado por instancias A3 (4 núcleos, 7GB de memoria) en la nube de Microsoft Azure; la tercera un cluster virtual de instancias A11 (16 núcleos, Intel Xeon E5-2670 @ 2.6Ghz, 112GB de memoria) también en Azure; y la cuarta un cluster virtual formado por instancias `m3.medium` (1 núcleo, 3.74GB de memoria) en la nube de AWS.

La figura 3 muestra el número de evaluaciones por segundo por núcleo (eval/s/core) para las implementaciones `asynPDE` y `SiPDE`, tanto en Plutón como en un cluster en Azure. Ésta es una buena métrica para evaluar las implementaciones porque no sólo incluye el tiempo de CPU de las evaluaciones si no que también tiene en cuenta las comunicaciones y otros sobrecostes de tiempo. A partir de estos resultados podemos concluir que:

- La implementación MPI tiene un valor de eval/s/core entre 2.1X y 2.5X veces mejor que la implementación con Spark.
- El número de eval/s/core de la implementación MPI en el cluster se reduce con el aumento del número de núcleos. Esto se debe a que a medida que el número de núcleos aumenta, también lo hace el sobrecoste de comunicaciones. Además, la computación de cada isla se reduce, empeorando el balance entre computación y comunicación. A la inversa, en la implementación de Spark, el sobrecoste de comunicaciones de la migración se mantiene constante, repartiendo los movimientos de datos entre los núcleos disponibles y, por lo tanto, presenta una mejor escalabilidad.

La figura 4 resume los resultados de los experimentos con las implementaciones de MR (mrPDE) y Spark (SiPDE) en el cluster. Se emplea una distribución de resultados (*beanplot*) para mostrar los tiempos de ejecución y la dispersión. De estos resultados podemos concluir lo siguiente:

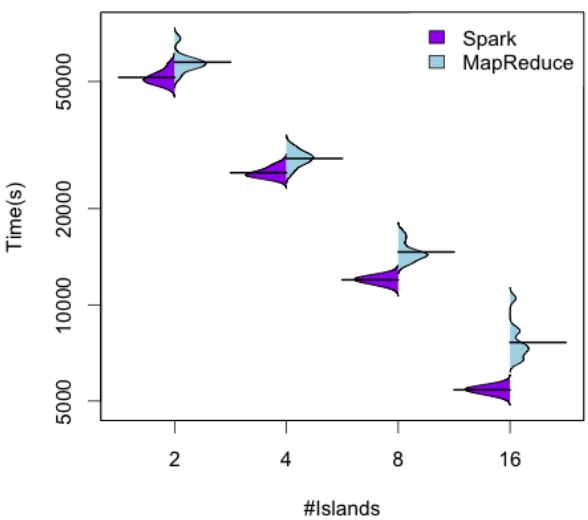


Fig. 4. Resultados de MR y Spark en Pluton.

- La implementación de MR tiene tiempos de ejecución y dispersión mayores que la implementación de Spark.
- La implementación de MR también reduce el tiempo de convergencia pero con una aceleración limitada.

Para evaluar el sobrecoste que está limitando la aceleración de MR, se ha medido, para un total de 8 iteraciones, el sobrecoste de cada iteración evolución-migración por separado. La figura 5 muestra la media y la desviación estándar de 10 ejecuciones independientes de cada experimento en Plutón y AWS. Para una descripción detallada del entorno de pruebas se puede consultar [10]. A partir de la figura podemos sacar las siguientes conclusiones:

- MR tiene un sobrecoste significativamente mayor y una dispersión más grande que Spark. Se debe destacar que, a pesar de que en un principio esto recomendaría no usarlo en favor de Spark, para problemas en los que los tiempos de ejecución dominan significativamente sobre el sobrecoste introducido por las iteraciones, MR sería competitivo con Spark aunque con peor escala-

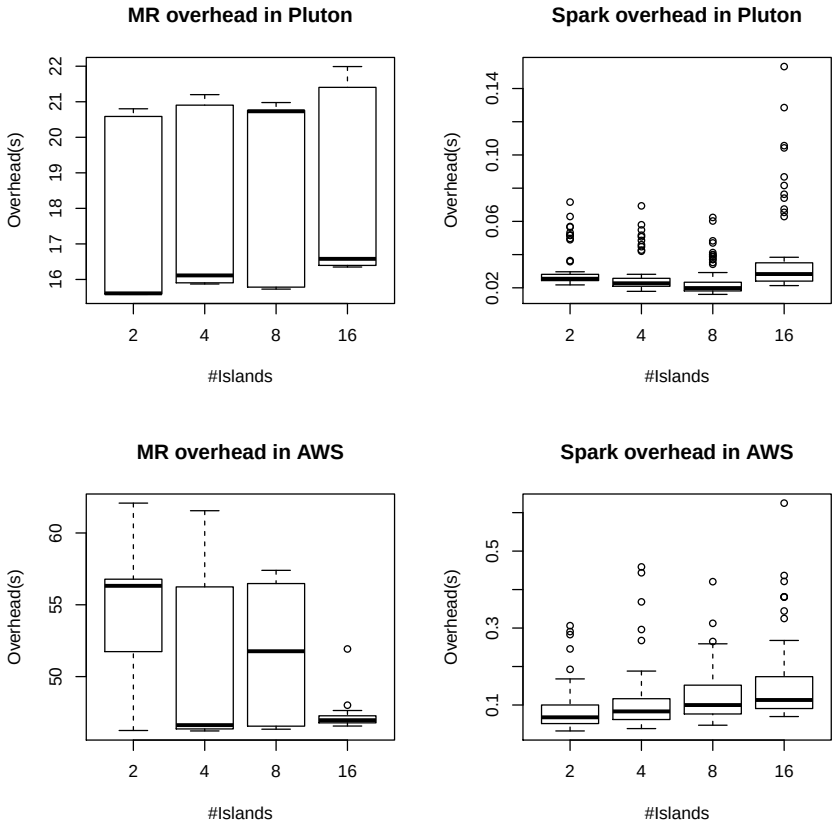


Fig. 5. Sobrecoste de MR y Spark por iteración evolución-migración

bilidad.

- Spark tiene mejor soporte para algoritmos iterativos que MR. Este era un resultado esperado, dado que Spark mantiene los resultados en memoria entre operaciones. Esto explica que las primeras iteraciones (los *outliers* en los diagramas de cajas) son las que más tiempo consumen en Spark, mientras que apenas hay diferencias entre iteraciones en MR.
- El sobrecoste de MR y Spark es mayor en AWS que en el cluster. Spark es entre 4x y 5x veces peor en AWS, y MR sobre 2.8x. También se debe destacar que, en sintonía con lo que se esperaba, el sobrecoste de Spark aumenta ligeramente cuando crece el número de nodos.

Las principales lecciones aprendidas de todos los experimentos realizados a lo largo de estos trabajos son:

- *Todas las implementaciones paralelas han mejorado el tiempo de convergencia del algoritmo secuencial.* Además, se obtienen aceleraciones superlineales porque, en el modelo de islas, la cooperación entre islas mejora el ritmo de convergencia.
- *La implementación MPI presenta los mejores resultados de convergencia (i.e, número de evaluaciones necesarias).* Esto es debido a las características inherentes de su modelo de programación.
- *La convergencia mejora con el número de islas.* El motivo es que la redistribución de la población, que se emplea en la estrategia de migración en la implementación con Spark, mantiene la diversidad cuando aumenta el número de islas.
- *La convergencia obtiene resultados similares en las plataformas cloud, pero el tiempo de ejecución empeora.* Los tiempos de ejecución son entre 2x y 3x veces más grandes en el cloud que en el cluster debido al sobrecoste de la virtualización y al uso de recursos no dedicados.

V. Conclusiones

En este paper hemos querido mostrar una visión general de las lecciones aprendidas, a través de nuestra experiencia, sobre las aproximaciones para implementar metaheurísticas paralelas en la nube. Usando la metaheurística de Evolución Diferencial, y un problema difícil en el campo de la biología de sistemas, como caso de prueba, hemos analizado tres paradigmas diferentes: MPI, MapReduce y Spark, en plataformas cloud públicas así como en un cluster local.

Los resultados muestran que, desde un punto de vista computacional, las implementaciones con MPI tienen mejor rendimiento que las basadas en Spark y Hadoop. Esto es debido a su lenguaje de programación de bajo nivel y al menor sobrecoste. Hadoop obtiene los peores resultados de rendimiento debido a su mayor sobrecoste causado por los mayores tiempos de inicialización de tareas y el acceso a los datos. Finalmente, Spark proporciona un mejor soporte para algoritmos iterativos reduciendo el sobrecoste entre

la primera y subsecuentes iteraciones. Nuestra experiencia en estos trabajos nos ha permitido entender que para obtener implementaciones eficientes es necesario remodelar los algoritmos existentes o crear otros nuevos que se adapten mejor a modelos de programación en la nube y puedan sacar ventaja de las capacidades que ofrecen estos entornos como elasticidad.

Agradecimientos

El presente trabajo ha sido financiado mediante el proyecto MINECO (Ministerio de Economía, Industria y Competitividad, Programa Estatal de Fomento de la Investigación Científica y Técnica de Excelencia) TIN2016-75845-P, y por la Xunta de Galicia bajo el programa de Consolidación y Estructuración de Unidades de Investigación Competitivas (ED431C 2017/04), cofinanciados con fondos FEDER de la UE.

Referencias

[1] Teodor Gabriel Crainic, "Parallel Meta-Heuristics and Cooperative Search," *CIRRELT*, vol. CIRRELT-2017-58, 2017.

[2] Enrique Alba, Gabriel Luque, and Sergio Nesmachnow, "Parallel metaheuristics: recent advances and new trends," *International Transactions in Operational Research*, vol. 20, no. 1, pp. 1–48, 2013.

[3] Roberto R. Expósito, Guillermo L. Taboada, Sabela Ramos, Juan Touriño, and Ramón Doallo, "Performance analysis of HPC applications in the cloud," *Future Generation Computer Systems*, vol. 29, no. 1, pp. 218–229, 2013.

[4] I. Sadooghi, J. H. Martin, T. Li, K. Brandstatter, K. Maheshwari, T. P. P. de Lacerda Ruivo, G. Garzoglio, S. Timm, Y. Zhao, and I. Raicu, "Understanding the Performance and Potential of Cloud Computing for Scientific Applications," *IEEE Transactions on Cloud Computing*, vol. 5, no. 2, pp. 358–371, 2017.

[5] Wei-Po Lee, Yu-Ting Hsiao, and Wei-Che Hwang, "Designing a parallel evolutionary algorithm for inferring gene networks on the cloud computing environment," *BMC systems biology*, vol. 8, no. 1, pp. 5, 2014.

[6] Abhishek Verma, Xavier Llora, David E Goldberg, and Roy H Campbell, "Scaling genetic algorithms using MapReduce," in *Ninth International Conference on Intelligent Systems Design and Applications, ISDA'09*. IEEE, 2009, pp. 13–18.

[7] Jeffrey Dean and Sanjay Ghemawat, "MapReduce: simplified data processing on large clusters," in *The 6th USENIX Symposium on Operating Systems Design and Implementation*, 2004.

[8] Matei Zaharia and et al., "Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing," in *The 9th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2012*, 2012.

[9] Paris Carbone, Asterios Katsifodimos, † Kth, Sics Sweden, Stephan Ewen, Volker Markl, Seif Haridi, and Kostas Tzoumas, "Apache flink^{TM}: Stream and batch processing in a single engine," *IEEE Data Engineering Bulletin*, vol. 38, 01 2015.

[10] Diego Teijeiro, Xoán C. Pardo, David R. Penas, Patricia González, Julio R. Banga, and Ramón Doallo, "Evaluation of parallel differential evolution implementations on mapreduce and spark," in *Euro-Par 2016: Parallel Processing Workshops*. 2017, pp. 397–408, Springer International Publishing.

[11] Diego Teijeiro, Xoan C. Pardo, Patricia González, Julio R. Banga, and Ramón Doallo, "Towards cloud-based parallel metaheuristics: A case study in computational biology with differential evolution and spark," *Journal of High Performance Computing Applications*, 2016.

[12] Patricia González, Xoan C. Pardo, David R. Penas, Diego Teijeiro, Banga Julio R., and Ramón Doallo, "Using the cloud for parameter estimation problems: Comparing spark vs mpi with a case-study," in *IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid '17)*, 2017, pp. 197–206.

[13] Kenneth V Price, Rainer M Storn, and Jouni A Lampinen, "Differential evolution a practical approach to global optimization," 2005.

[14] S. Das and P.N. Suganthan, "Differential evolution: A survey of the state-of-the-art," *IEEE Transactions on Evolutionary Computation*, vol. 15, no. 1, pp. 4–31, 2011.

[15] Diego Teijeiro, Xoán C. Pardo, Patricia González, Julio R. Banga, and Ramón Doallo, *Applications of Evolutionary Computation: 19th European Conference, EvoApplications 2016, Proceedings, Part II*, chapter Implementing Parallel Differential Evolution on Spark, pp. 75–90, 2016.

[16] J.C.W. Locke, A.J. Millar, and M.S. Turner, "Modelling genetic networks with noisy and varied experimental data: the circadian clock in arabidopsis thaliana," *Journal of Theoretical Biology*, vol. 234, no. 3, pp. 383–393, 2005.

BETi: Sistema para la gestión y procesamiento de datos masivos LiDAR

David Deibe[1], Margarita Amor[1] y Ramón Doallo[1]

Resumen— **Las nubes de puntos LiDAR son una de las fuentes de información geográfica más valiosas de la actualidad. No obstante, las características especiales de LiDAR han convertido a esta tecnología en un verdadero reto a la hora de desarrollar aplicaciones que sea capaces de almacenar, gestionar, visualizar o analizar conjuntos de datos masivos pertenecientes a este tipo de información. En este trabajo presentamos el sistema BETi (Big data for the Exploration of Terrain Information), un sistema escalable y eficiente para el almacenamiento y la computación distribuida de datos masivos LiDAR. El principal objetivo de este sistema el el de dar soporte a aplicaciones, principalmente web.**

Palabras clave— **LiDAR, big data, almacenamiento, computación, visualización**

I. Introducción

En la actualidad, la tecnología LiDAR (*Light detection and ranging*) es considerada como una de las fuentes de información geográfica más valiosas, ya que es capaz de proporcionar datos geoespaciales de muy alta resolución en forma de nubes de puntos altamente detalladas. Entre las distintas aplicaciones y usos de los datos LiDAR pueden encontrarse estudios tan diversos como el análisis de volcanes [1] o la clasificación de entornos rurales [2], entre otros muchos. Además, la combinación de LiDAR con otros tipos de datos de detección remota puede proporcionar beneficios adicionales en varios contextos [3]. Una extensa lista de aplicaciones adicionales puede encontrarse en [4], mientras que en [5] se presenta una extensa revisión sobre los logros y ventajas del uso de la tecnología LiDAR.

Debido a las extraordinarias grandes cantidades de datos que se pueden ser recolectadas, en muchos casos superando los terabytes o incluso petabytes de almacenamiento, las nubes de puntos LiDAR son bien conocidas por ser representar desafío a la hora de desarrollar software para su almacenamiento, procesamiento, visualización o análisis, llevando a los investigadores a buscar constantemente nuevas formas de lidiar de manera eficiente con este tipo de información geoespacial. Así, en centros GIS, instituciones gubernamentales o en cualquier otro tipo de grupo que trabaje con altas tasas de recopilación de datos, el uso de sistemas escalables y eficientes para el manejo de datos LiDAR supone un requisito fundamental.

Parte de la investigación desarrollada por el Grupo de Arquitectura de Computadores (GAC) de la Universidade da Coruña (UDC) en estos últimos años ha girado en torno al desarrollo y mejora de diversos algoritmos, técnicas y sistemas, ubicados en diferentes etapas críticas a lo largo de las distintas etapas de procesamiento de datos LiDAR. Las contribuciones más importantes abarcan desde métodos eficientes para la consulta de datos georreferenciados hasta herramientas de medición geoespacial para campos científicos específicos, soluciones para la movilidad y disponibilidad de software LiDAR [6], algoritmos de compresión de datos sin pérdida, estructuras de datos no redundantes para el soporte de técnicas multiresolución [7], así como también sistemas altamente escalables para el almacenamiento [8] y computación distribuidas [9] de grandes volúmenes de datos, todo ello a través del uso de los más actuales y novedosos paradigmas de programación y tecnologías *big data*. Así, en este trabajo presentamos la plataforma BETi (Big data for the Exploration of Terrain Information), como la integración de todos los elementos que han sido desarrollados estos últimos años y que sirven así para trabajar de forma completa con datos masivo LiDAR tomados desde aeronaves.

El resto de este trabajo está estructurado de la siguiente manera: la tecnología de datos LiDAR se presenta brevemente en la Sección II. En la Sección III se muestran las principales características del sistema BETi. Durante la Sección IV se muestran los resultados, mejoras y ventajas que ofrece el sistema. Finalmente, en la Sección V se presentan las conclusiones.

II. Tecnología LiDAR

LiDAR es un método de medición que emplea pulsos de láser para obtener distancias entre un sensor y un objetivo determinado. Las diferencias en los tiempos de retorno de los pulsos proporcionan información 3D sobre las superficies en las que los pulsos son reflejados. Toda la información recopilada se presenta en forma de nubes de puntos 3D. Además de las coordenadas espaciales (x, y, z), se puede recopilar información adicional tal como la intensidad del pulso, el color RGB o la clasificación (construcción, vegetación, agua, etc.), entre otros.

Con el fin de escanear grandes extensiones de tierra, los sensores LiDAR pueden ser montados casi en cualquier tipo de vehículo, desde vehículos aéreos no tripulados (UAV) hasta aviones, automóviles o barcos; sin embargo, los sensores también pueden montarse en estructuras estáticas para capturar escenas 3D en interiores o áreas al aire libre de dimensiones reducidas. La precisión lograda por LiDAR permite crear representaciones 3D muy precisas de cualquier tipo de superficie objetivo, incluidos ele-

[1]Grupo de Arquitectura de Computadores, CITIC, Universidade da Coruña (UDC), e-mails: david.deibe@udc.es, margarita.amor@udc.es y ramon.doallo@udc.es

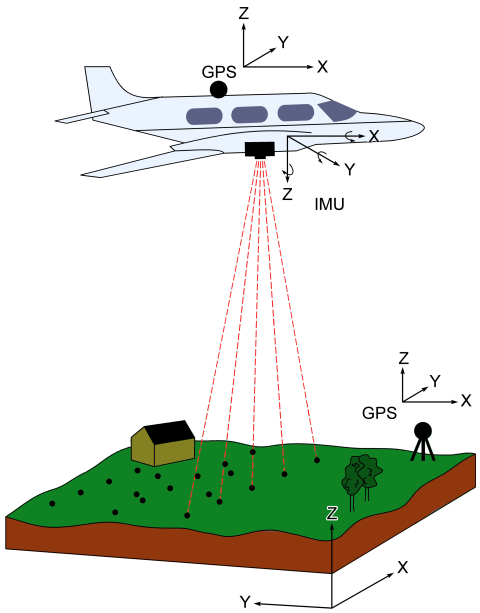


Fig. 1. Land surveying using ALS. Source: [10].

mentos de gran complejidad o elementos que son difíciles de identificar o capturar a través de otros métodos topográficos; por ejemplo, superficies bajo las copas de los árboles o líneas eléctricas de muy bajo diámetro.

III. BETi (Big data for the Exploration of Terrain Information)

La investigación sobre aplicaciones y soluciones para la gestión y manejo de datos masivos LiDAR llevado a cabo por el GAC en la UDC, comenzó con el desarrollo de aplicaciones para la visualización de nubes de puntos y la toma de mediciones sobre las propias imágenes, ya que estas son dos de las herramientas más fundamentales para la exploración e interpretación de los datos LiDAR. El diseño de este tipo de software no debe ser ajeno a las necesidades y requisitos de sus usuarios finales, ya que, en muchas ocasiones, su productividad y la calidad de su trabajo están directamente relacionadas con la calidad del diseño del software y sus funcionalidades. Es por ello que se desarrollaron dos propuestas (GVLiDAR [6] y ViLMA [7]) con el objetivo de cubrir necesidades de visualización y renderizado diferentes, centrando no obstante el desarrollo de ambas en torno a la flexibilidad y la movilidad, los flujos de trabajo adecuados y las herramientas de medición para campos científicos específicos, por ejemplo, las ciencias agroforestales.

- **Flexibilidad**: WebGL [11] fue la interfaz de programación elegida para el desarrollo de los motores de renderizado, ya que a través de la implementación de una solución de visualización web, fue posible desvincularse de cualquier sistema operativo (OS) o dispositivo en particular, otorgando acceso instantáneo a cualquier usuario desde cualquier ubicación con el único requisito de ejecutar un navegador web compatible con WebGL. El desarrollo se concibió con un enfoque orientado al servicio, por lo que la aplicación y los datos se almacenarán en un servidor remoto y los clientes acceden a estos según sea necesario. La arquitectura cliente-servidor (ver Figura 2) en conjunción con el enfoque de servicio otorga las aplicaciones que incorporan estos motores de puntos un gran nivel de movilidad y flexibilidad.
- **Flujo de trabajo adecuado**: Aunque algunos conjuntos de datos pueden llegar a representar superficies de varios kilómetros cuadrados, en muchas ocasiones, el trabajo de los usuarios requiere el uso de regiones específicas dentro un conjunto mayor, lo que lleva en muchas ocasiones a desperdiciar muchos y valiosos recursos computaciones. Así pues, mediante técnicas de hashing espacial y la posibilidad de definir regiones de interés personalizadas por el usuario,

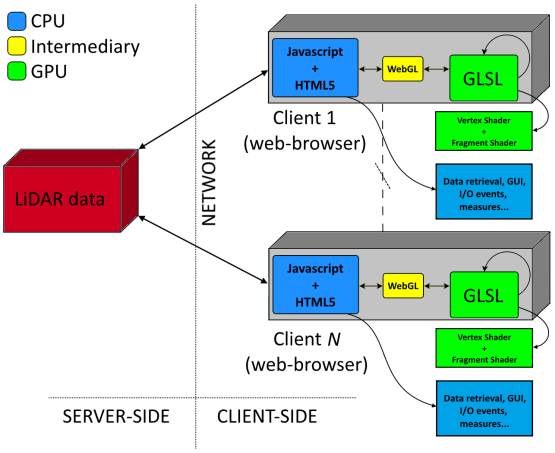


Fig. 2. Patrón cliente-servidor seguido para el diseño de los visualizadores web. En la imagen se detallan únicamente los componentes más importantes del lado cliente.

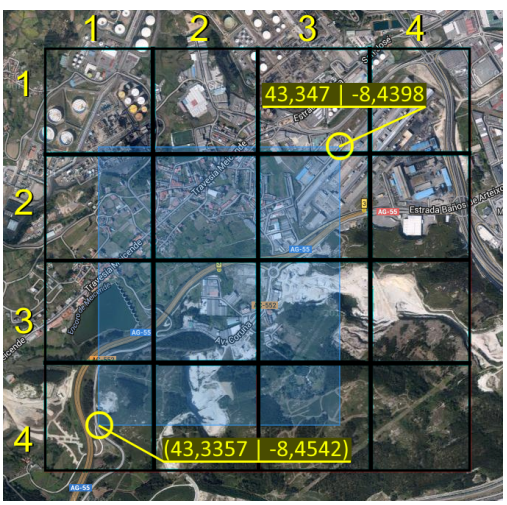
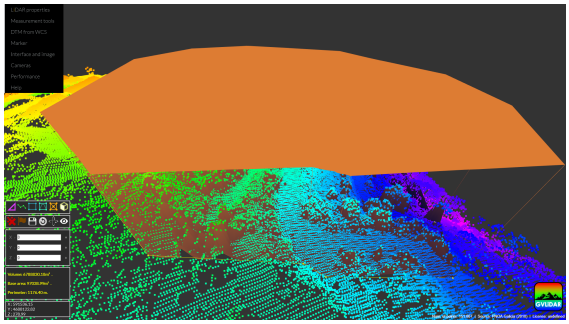
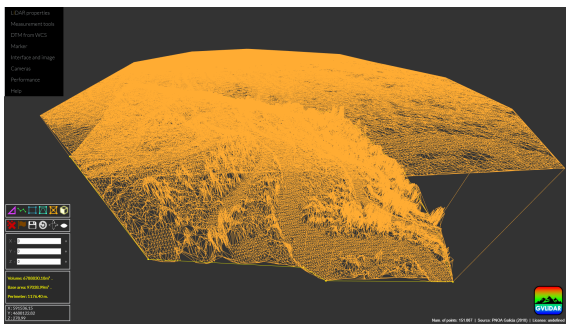


Fig. 3. Hashing espacial utilizado para la obtención rápida y eficiente de datos desde el cliente. Las coordenadas que delimitan la región de interés del usuario (sub-área interior de la imagen) pueden ser transformadas en las posiciones de una celda perteneciente a una malla regular la cual tiene una equivalencia con ficheros de datos almacenados en el servidor.



(a)



(b)

Fig. 4. Herramienta de medición de volúmenes complejos realizada directamente sobre la nube de puntos. 4(a) Nube con volumen sólido. 4(b) Volumen en formato malla de triángulos.

se implementaron rápidas y eficientes consultas de datos basadas en restricciones espaciales, acercando la funcionalidad de los visores al flujo de trabajo de los usuarios (ver Figura 3).

- **Herramientas de medición específicas**: A través de JavaScript (JS) [12] y WebGL, diversas herramientas fueron implementadas para permitir a los usuarios la toma de mediciones geoespaciales directamente sobre las imágenes de las nubes de puntos desde el lado cliente. Estas herramientas, diseñadas en función de las necesidades específicas de campos científicos como las ciencias agroforestales, proporcionaban a los usuarios un rango más amplio y útil de herramientas en comparación con lo que ofrecían otras soluciones de visualización. Como ejemplo de estas herramientas pueden encontrarse herramientas para la medición de superficies en fachadas u otras mucho más complejas, como los volúmenes de base irregular, el contorno poligonal y la parte superior proyectada (ver Figura 4).

Como ya se comentó al principio de esta sección, el desarrollo de las dos aplicaciones de visualización se centró en la consecución de los objetivos arriba listados siguiendo enfoques diferentes en lo que se refiere al renderizado de las nubes de puntos. Las características particulares de cada una de ellas son detalladas a continuación:

GVLiDAR [6] se diseñó con el objetivo de mostrar nubes de puntos de resolución completa, maximizando la información en pantalla para maximizar la calidad y precisión de los análisis visuales y las herramientas de medición. Para su diseño, no se emplearon ni técnicas multiresolución ni ningún otro tipo de mejora visual. WebGL fue un elemento clave para lograr estos objetivos y, al mismo tiempo, lograr un buen rendimiento, no solo aprovechando el poder computacional de la unidad de procesamiento gráfico (*graphics processing unit* o GPU) del cliente, sino también mediante la implementación de técnicas de optimización gráfica como el *view-frustum-culling* o un método de *occlusion-culling* especialmente implementado para esta propuesta [13].

Por su parte, con ViLMA [7] se exploraron nuevas direcciones en el uso de técnicas multiresolución *out-of-core*. Como resultado, se desarrolló una nueva estrategia de visualización respaldada por un método de ordenación de puntos sin redundancia de datos llamado *Hierarchically Layered Tiles* (HLT), y una estructura de tipo árbol llamada *Tile Grid Partitioning Tree* (TGPT). Estas estructuras de datos se diseñaron con el objetivo de obtener máxima eficiencia durante el renderizado de puntos evitando la redundancia de datos siempre asociada a la creación y administración de diferentes niveles de detalle en los sistemas multiresolución. La idea principal detrás de estas estructuras era la de reorganizar y almacenar los puntos de cada nube en capas dentro de una serie de celdas, de tal modo que ningún punto fuese repetido a través de las diferentes capas. Estas celdas divididas en capas sirven como piezas de puzle que permiten calcular y crear diferentes niveles de detalle en tiempo de ejecución mediante la fusión de dos o más de estas piezas, todo ello en base a la potencia de cálculo de la GPU del cliente y los movimientos de cámara de la escena. Estas estructuras sin redundancia de datos permiten obtener niveles muy bajos de consumo de RAM y memoria de vídeo (VRAM), ayudan a minimizar los requisitos de almacenamiento del servidor y del cliente y reducen el tráfico de red.

A. Compresión de datos sin pérdida

Todos los datos LiDAR manejados por el sistema BETi son comprimidos sin pérdida de información empleando nuestro propio formato de compresión llamado *LZ* [7]. Durante la compresión de datos se realizan tres tareas principales; limpieza de datos, codificación delta y compresión GZIP:

- **Limpieza de datos**: En esta primera tarea se simplifica y reorganiza la información almacenada para cada punto. Las propiedades no necesarias para la visualización y procesamiento, como por ejemplo el rango de ángulo de escaneo, pueden no incluirse en los archivos *LZ*, entre otras propiedades seleccionadas. Otras propiedades, como la información de Intensidad, Clasificación o Devolución, se adaptan o modifican para optimizar su tamaño, teniendo en cuenta su función dentro de la aplicación.
- **Codificación delta**: La codificación delta, es un método para almacenar datos en forma de diferencias o deltas entre datos secuenciales. Las

propiedades de un punto dado se derivan de las propiedades de su predecesor más una serie de diferencias. Una o más máscaras de bytes (dependiendo de la cantidad de propiedades por punto que se desee almacenar) son utilizadas por cada punto para especificar si ha habido, o no, un cambiado en sus propiedades respecto a las del punto anterior. Si determinada propiedad ha cambiado, la máscara también informa a cerca de la longitud del byte de la diferencia (delta) que debe utilizarse.

- **Compresión GZIP**: Si bien los métodos de compresión de uso general no son la mejor opción para los datos LiDAR, cuando se aplican junto con técnicas como la codificación delta, pueden obtenerse archivos altamente comprimidos y sin pérdidas. El software GZIP [14], emplea un método de compresión basado en el algoritmo DEFLATE, ampliamente utilizado en aplicaciones web y otros entornos de red. GZIP no solo logra excelentes *ratios* de compresión, sino que también es compatible, de manera nativa, con los principales navegadores web del mercado. De este modo, todas las tareas relativas a la descompresión de ficheros comprimidos mediante GZIP son llevadas a cabo de manera automática y eficiente por parte del navegador. La aplicación de visualización web o la herramienta que reciba los datos LiDAR, solo tendrá que realizar la decodificación delta para obtener los datos originales de los puntos.

B. Almacenamiento distribuido mediante big data

La mayoría de aplicaciones web de tipo cliente-servidor para la visualización de datos masivos de LiDAR, han confiado para la operatividad de su lado servidor en soluciones clásicas como Apache HTTP Server. Este tipo de soluciones han sido utilizadas durante años ofreciendo una gran robustez, rendimiento y seguridad. Sin embargo, carecen de muchas ventajas ofrecidas de forma nativa por la mayoría de tecnologías *big data* para almacenamiento distribuido, como son la baja latencia y alto *throughput* de datos, la escalabilidad horizontal, la alta disponibilidad a través de la replicación de datos, la distribución de datos en varias máquinas o la integración de soluciones de computación distribuida. En la Figura 5 puede observarse la estructura general de una aplicación web orientada a la visualización de datos LiDAR mediante multiresolución. La imagen principal (los dos tercios superiores) representa un diseño que sigue el patrón cliente-servidor. En este diseño la caja etiquetada con una letra **A** marca los componentes que siguen un enfoque clásico mediante el uso de una sola máquina montando un software Apache HTTP Server. La adaptación de este sistema a un entorno *big data* requiere la sustitución de los elementos contenidos en **A** por aquellos indicados en la caja etiquetada como **B**. Las ventajas de seguir un enfoque *big data* en contextos de información geoespacial han sido

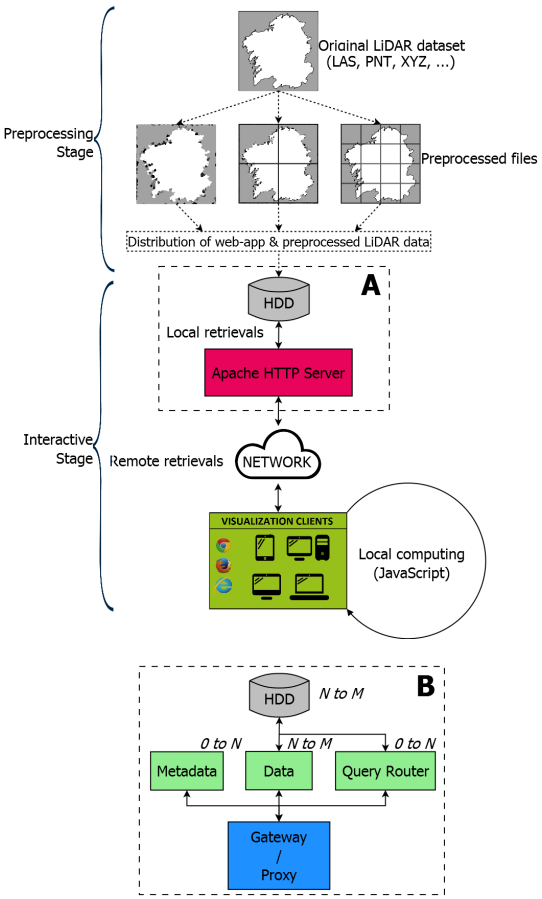


Fig. 5. Estructura general de una aplicación web para la visualización de datos LiDAR. La caja etiquetada con una letra **A** marca los componentes que siguen un enfoque clásico dentro del patrón cliente-servidor. La adaptación de este sistema a un entorno *big data* requiere la sustitución de los elementos contenidos en **A** por aquellos contenidos en **B**.

analizadas en diversos trabajos [15], algunos de ellos centrándose en los beneficios obtenidos en el campo específico de LiDAR [16].

Para lograr la integración de las diferentes aplicaciones LiDAR anteriormente presentadas en un entorno de almacenamiento *big data*, logrando con ello además adquirir todas las ventajas asociadas al uso de este tipo de tecnologías, se llevaron a cabo una serie de análisis y experimentos para determinar, de entre todas las tecnologías y paradigmas disponibles, cuál era la más adecuada para el caso de uso que nos atañía. Para ello, se tomaron cuatro de las tecnologías de almacenamiento más utilizadas y con un desarrollo más maduro, siendo estas: HDFS [17], MongoDB [18], Cassandra [19] y Redis [20]. Con ellas, se demostró que la visualización en tiempo real de conjuntos de datos LiDAR masivos a través del software basado en la web puede ser soportada de manera eficiente por dichas tecnologías sin ningún inconveniente o penalización en el rendimiento o la experiencia del usuario, al tiempo que se obtienen todos de las ventajas asociadas al uso de *big data*, como la escalabilidad o la tolerancia a fallos [8]. En concreto, Cassandra fue la tecnología elegida como la más idónea para sustituir a Apache HTTP Server.

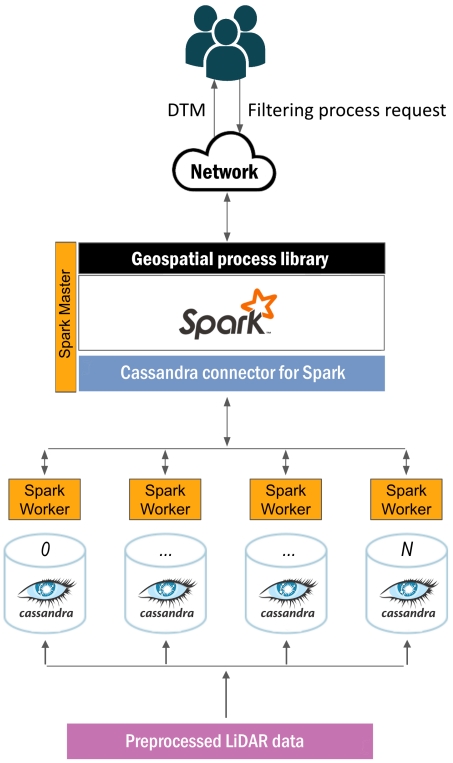


Fig. 6. Estructura general de la capa computacional del sistema BETi.

Además, con miras al futuro, los sistemas que utilicen ese tipo de tecnologías ya estarían preparados para incorporar tecnologías de procesamiento distribuido a gran escala como Spark, Storm o Flink.

C. Computación distribuida mediante big data

Los modelos de elevación GIS, como el *digital surface model* (DSM) o el *digital terrain model* (DTM), son uno de los productos más importantes y valiosos derivados de las nubes de puntos LiDAR, ya que estos modelos tridimensionales (3D) de tipo raster son el elemento fundamental para muchos procesos geoespaciales, por ejemplo, estimación de biomasa [21] o extracción de características lineales [22]. La calidad de estos modelos de elevación está fuertemente relacionada con la clasificación precisa de los puntos LiDAR en las categorías de terreno o no-terreno. Los tiempos de cómputo y los requisitos de almacenamiento involucrados en este tipo de clasificación pueden convertirse en un problema crítico en contextos con altas tasas de recolección de datos, como los mencionados en secciones anteriores. Teniendo en cuenta los volúmenes masivos de datos que deben almacenarse y gestionarse en tales contextos, una sola máquina y el uso de soluciones software convencionales pueden sufrir problemas importantes como la falta de escalabilidad y disponibilidad, bajos rendimientos y altos niveles de latencia durante la ejecución de tareas computacionalmente complejas.

Finalmente el sistema BETi se completó con el desarrollo e integración de un sistema para la ejecución de procesos geoespaciales sobre conjuntos de datos LiDAR masivos mediante tecnologías *big data* para computación distribuida [9]. Como caso de estudio inicial, la investigación se centró en la obtención rápida de DTMs siguiendo el análisis y las conclusiones presentadas en la subsección anterior, la distribución de datos se realizó mediante Cassandra [19], mientras que para la distribución de cómputos y tareas se empleó Spark [23] debido a su versatilidad y diseño orientado a procesamiento por lotes. La Figura 6 muestra la estructura general del sistema BETi con especial detalle en su lado servidor.

Mediante el uso de estas dos tecnologías fue posible reducir enormemente el tiempo requerido para procesar grandes extensiones de nubes de puntos aéreos en comparación con los enfoques de una sola máquina y las herramientas de escritorio actualmente disponibles.

IV. Resultados

En esta sección se presentan los resultados y análisis relativos al rendimiento, funcionalidades y calidad que BETi es capaz de proporcionar desde el punto de vista la visualización, la compresión de datos y el almacenamiento y computación distribuidas. La Tabla I muestra las características de los principales datasets utilizados en esta sección. Las máquinas cliente utilizadas para ejecutar los visualizadores se describen en la Tabla II así como también los nodos usados para la instalación del software en el lado servidor.

TABLA I
Principales datasets utilizados para los análisis.

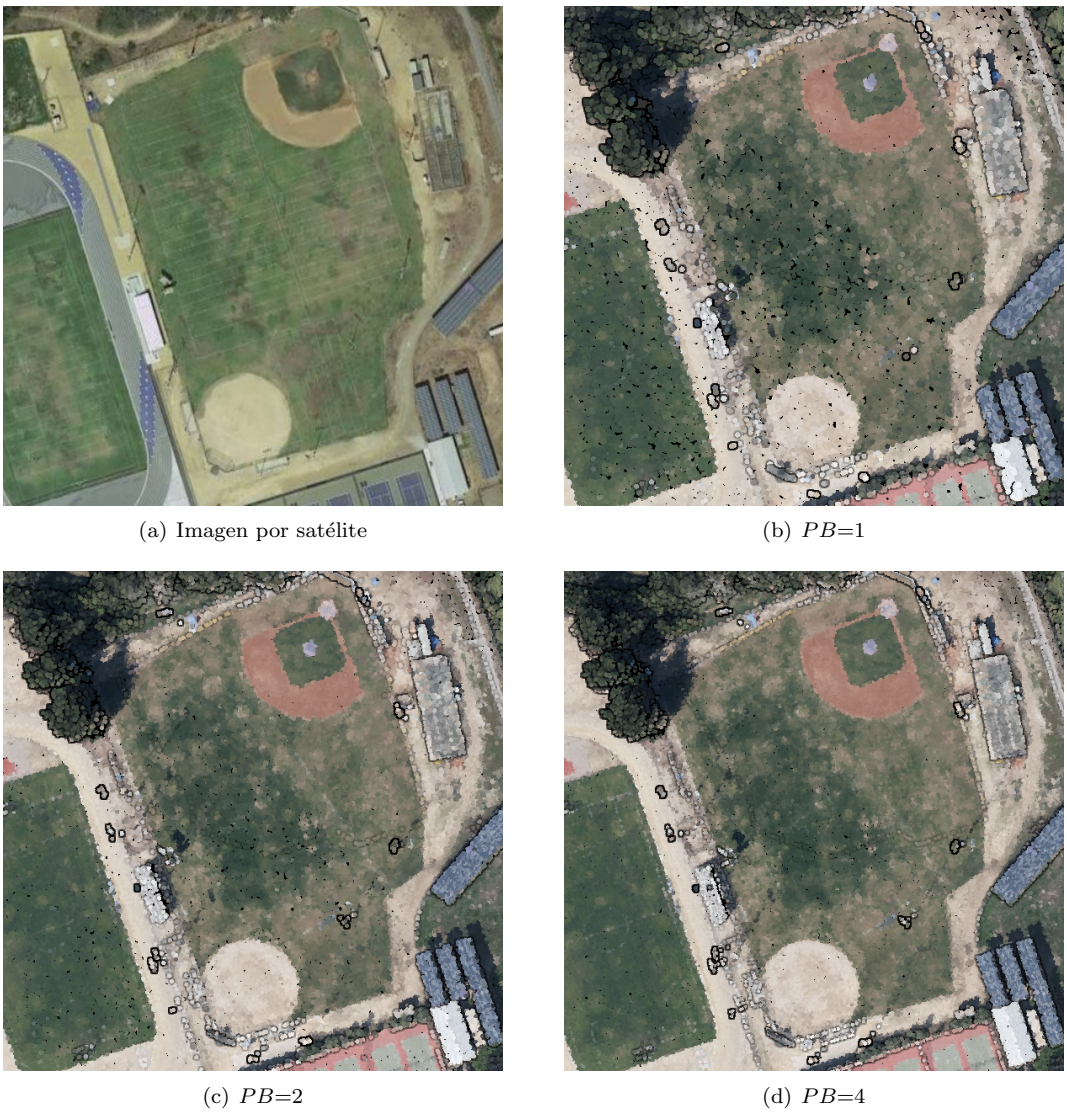
Dataset	Puntos (billones)	Tamaño original (GB)
Galicia PNOA	28	802
Guitiriz	1	37
San Simeon	17.7	561
Volcano	0.55	14.6

TABLA II
Características de las máquinas cliente y los nodos para servidor.

Platform	O.S.	CPU	GPU	RAM (GB)	VRAM (GB)	Bandwidth (Mbps)
Client PC	Windows 7	Intel Core i7 4790	GeForce Titan X	32	12	90 (Wired)
Client Mobile	Android 7.0	Tegra K1	Tegra K1	2 (Unified)	-	65 (Wi-Fi)
Server Node	CentOS 6.9	2 x Intel Xeon E5-2650 v2	-	64	-	InfiniBand

La Figura 7 muestra una comparativa entre una imagen por satélite y las nubes de puntos renderizadas por BETi utilizando ViLMA como visualizador utilizando distintas cantidades de puntos (point budget o PB). Este PB representa una cantidad fija de puntos que puede ser definida por el usuario y que establece el grado de detalle con el que se van a representar las diferentes áreas de la nube de puntos bajo el sistema multiresolución.

Para evaluar la calidad de compresión de nuestro formato LZ, se ha realizado una pequeña comparativa entre este y el formato LAZ, el más utilizado



(a) Imagen por satélite (b) PB=1

(c) PB=2 (d) PB=4

Fig. 7. Vista sobre un campo de béisbol renderizado por *BETi* utilizando diferentes PB: (a) Imagen real por satélite. (b)-(d) Nubes de puntos renderizadas utilizando 1, 2 y 4 millones de puntos respectivamente.

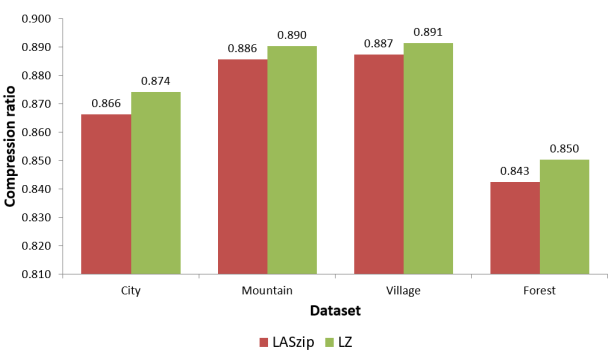


Fig. 8. Comparación entre los *ratios* de compresión de los formatos LZ y LASzip (LAZ).

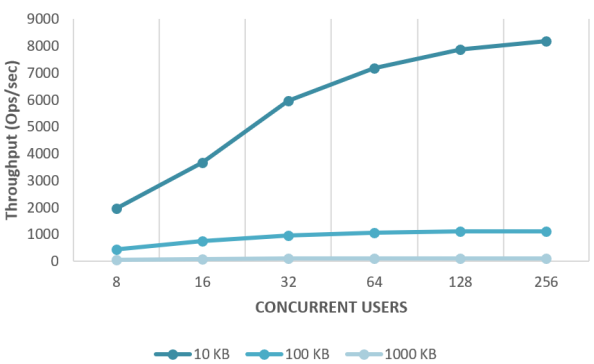


Fig. 9. *Throughput* observado en Cassandra utilizando diferentes tamaños de ficheros y con diferente número de usuarios concurrentes.

en el sector LiDAR. En la Figura 8 puede observarse como LZ obtiene mejores *ratios* de compresión $(1 - (Size_{compressed} \ / \ Size_{uncompressed}))$ que LAZ para todos los diferentes tipos de nube de puntos. El formato LZ no solamente nos proporciona mejores *ratios* de compresión si no que nos permite mayor libertad de acción y programación a la hora de desarrollar los distintos módulos y componentes de BETi, ajustando el formato y características de los ficheros a las necesidades concretas de cada etapa de procesado.

Como se comentó en secciones anteriores, Cassandra fue la tecnología *big data* elegida para encargarse del almacenamiento distribuido en el sistema BETi.

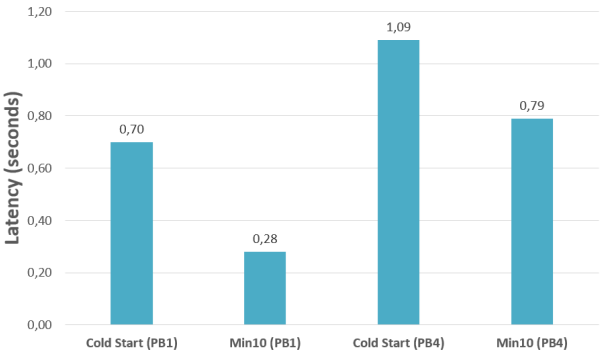


Fig. 10. Latencia observada en Cassandra utilizando diferentes tamaños de ficheros y con diferente número de usuarios concurrentes.

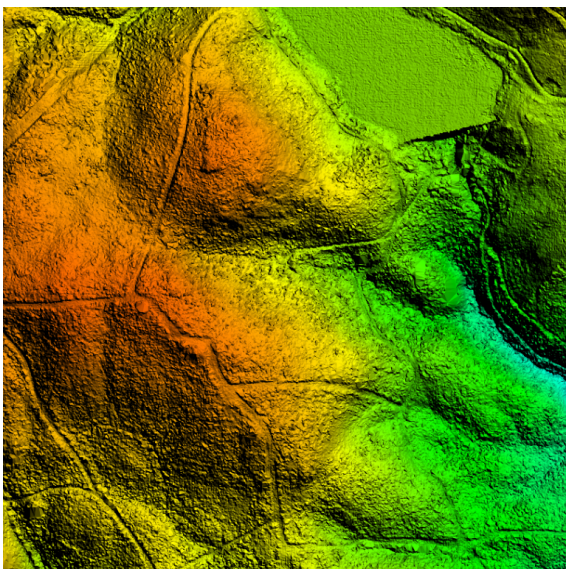


Fig. 11. DTM obtenido por el sistema BETi a partir de una nube de puntos de alta densidad.

En la Figura 9 puede observarse el *throughput* que obtiene Cassandra 3.9.0 para soportar el envío de datos al visor ViLMA utilizando distintos tamaños de fichero LiDAR y diferente número de usuarios concurrentes. Además, en la Figura 10 se muestran las latencias que se obtienen al pedir una nube de puntos desde ViLMA utilizando diferentes PB y midiéndose los tiempos en *cold start* (con el sistema recién arrancado) y tomando el valor mínimo tras 10 peticiones de datos consecutivas.

Mediante el uso de Spark 2.4.0 y Cassandra 3.11.3 se analizó el rendimiento del sistema BETi para el filtrado de nubes de puntos en las categorías de terreno y no-terreno con el objetivo de obtener en última instancia MDTs de gran calidad, obteniendo un tiempo medio de 3 horas y media para el filtrado de Galicia (28.000 millones de puntos). Un ejemplo de la calidad de los MDTs que el sistema es capaz de obtener puede observarse en la Figura 11. Información más detallada acerca del funcionamiento y rendimiento de BETi para el filtrado de puntos y otros proceso geoespaciales podrá encontrarse en el trabajo enviado a [9], ya que esta parte todavía se encuentra en desarrollo.

V. Conclusiones

BETi es el resultado de la integración de diversas investigaciones y módulos computacionales desarrollados en el GAC a lo largo de varios años. Así, BETi ofrece varias soluciones para la visualización eficiente de nubes de puntos, tanto para nubes de resolución completa como para nubes renderizadas bajo multiresolución, siempre logrando altos niveles de rendimiento en términos de FPS y calidad visual. Además, las distintas soluciones ofrecen la posibilidad de realizar mediciones directamente sobre las nubes de puntos ofreciendo un flujo de trabajo, una movilidad y flexibilidad a la altura de las necesidades de los usuarios más especializados en el sector LiDAR. BETi se presenta como un sistema altamente escalable, perfectamente preparado para almacenar y gestionar cualquier volumen de datos de manera eficiente y distribuida. Mediante el uso de tecnologías *big data*, en este caso Cassandra y Spark, se logró obtener una solución de alta capacidad para la distribución y paralelización a gran escala de todo tipo de procesos geoespaciales.

Como trabajo futuro, se pretende aumentar el número de procesos geoespaciales que BETi es capaz de ofrecer.

Agradecimientos

Esta investigación ha sido financiada por la Xunta de Galicia bajo el Programa de Consolidación para Grupos de Referencia Competitiva, co-financiado por los fondos ERDF de la Unión Europea [Ref. ED431C 2017/04]; bajo el Programa de Consolidación para Unidades de Investigación Competitivas, co-financiado por los fondos ERDF de la Unión Europea [Ref. R2016/037]; por la Xunta de Galicia (Centro Singular de Investigación de Galicia, con acreditación 2016/2019) y por los fondos ERDF de la Unión Europea [Ref. ED431G/01]; y por el Ministerio de Economía y Competitividad del gobierno de España y los fondos ERDF de la Unión Europea [TIN2016-75845-P].

Todos los conjuntos de datos que aparecen en este trabajo pertenecen a:

- *Guitiriz*: Aportado por el Laboratorio do Territorio (LaboraTe) [24].
- *Galicia*: Ficheros obtenidos del repositorio LiDAR-PNOA, del ©Instituto Geográfico Nacional de España [25].
- *PG&E Diablo Canyon Power Plant (DCPP), San Simeon, CA Central Coast y Sunset Crater Volcano National Monument, AZ*: Datos obtenidos desde la web OpenTopography [26].

Referencias

[1] G. Kereszturi, J. Procter, S.J. Cronin, K. Nemeth, M. Bebbington, and J. Lindsya, "LiDAR based quantification of lava flow susceptibility in the City of Auckland (New Zealand)," *Remote Sensing of Environment*, vol. 125, pp. 198–213, 2012.

[2] S. Buján, E. González-Ferreiro, L. Barreiro-Fernández, I. Santé, E. Corbelle, and D. Miranda, "Classification of rural landscapes from low-density lidar data: is it the-

oretically possible?," *International Journal of Remote Sensing*, vol. 34, no. 16, pp. 5666–5689, 2013.
[3] J. Zhang, "Multi-source remote sensing data fusion: status and trends," *International Journal of Image and Data Fusion*, vol. 1, no. 1, pp. 5–24, 2010.
[4] Paolo Tarolli, "High-resolution topography for understanding earth surface processes: Opportunities and challenges," *Geomorphology*, vol. 216, pp. 295–312, 2014.
[5] J. J. Roering, B. H. Mackey, J. A. Marshall, K. E. Sweeney, N. I. Deligne, A. M. Booth, A. L. Handwerger, and C. Cerovski-Darriau, "You are HERE: Connecting the dots with airborne LiDAR for geomorphic fieldwork," *Geomorphology*, vol. 200, pp. 172–183, 2013.
[6] David Deibe, Margarita Amor, Ramón Doallo, David Miranda, and Miguel Cordero, "GVLiDAR: An interactive web-based visualization framework to support geospatial measures on lidar data," *International Journal of Remote Sensing.*, vol. 38, no. 3, pp. 827–849, 2017.
[7] David Deibe, Margarita Amor, and Ramón Doallo, "Supporting multi-resolution out-of-core rendering of massive lidar point clouds through non-redundant data structures," *International Journal of Geographical Information Science*, vol. 33, no. 3, pp. 593–617, 2019.
[8] D. Deibe, M. Amor, and R. Doallo, "Big data storage technologies: a case study for web-based lidar visualization," in *2018 IEEE International Conference on Big Data (Big Data)*, Dec 2018, pp. 3831–3840.
[9] David Deibe, Margarita Amor, and Ramón Doallo, "Big data geospatial processing for aerial lidar datasets," *Computers, Environment and Urban Systems*, 2019, Submitted.
[10] Marek9134, "Lidar-i lend," `https://commons.wikimedia.org/wiki/File:LiDAR-i_lend.gif`, Colour, new labels and text by David Deibe, https://creativecommons.org/licenses/by-sa/3.0/legalcode. Accessed on 02/19/2019.
[11] Tony Parisi, *WebGL: Up and Running*, O'Reilly Media, Inc., 1st edition, 2012.
[12] David Flanagan, *JavaScript: The Definitive Guide*, O'Reilly Media, Inc., 6st edition, 2011.
[13] Tomas Akenine-Moller, Eric Haines, and Naty Hoffman, *Real-Time Rendering*, A. K. Peters, Ltd., 3rd edition, 2008.
[14] GNU Project - Free Software Foundation, "Gzip," `https://www.gnu.org/software/gzip/`, Accessed on 01/22/2019.
[15] Songnian Li, Suzana Dragicevic, Francesc Antón Castro, Monika Sester, Stephan Winter, Arzu Coltekin, Christopher Pettit, Bin Jiang, James Haworth, Alfred Stein, and Tao Cheng, "Geospatial big data handling theory and methods: A review and research challenges," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 115, pp. 119 – 133, 2016.
[16] Jan Boehm, "File-centric organization of large lidar point clouds in a big data context," in *IQmulus First Workshop on Processing Large Geospatial Data*, 2014, pp. 69–76.
[17] The Apache Software Foundation, "Apache hadoop 3.0.0 - hdfs architecture," `http://hadoop.apache.org/docs/r3.0.0/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html`, Accessed on 01/22/2019.
[18] MongoDB, Inc, "Mongodb," `http://www.mongodb.com/`, Accessed on 01/22/2019.
[19] The Apache Software Foundation, "Apache cassandra," `http://cassandra.apache.org/`, Accessed on 01/22/2019.
[20] Redis Labs, "Redis," `http://redis.io/`, Accessed on 01/22/2019.
[21] Henrik Persson, Jörgen Wallerman, Håkan Olsson, and Johan E.S. Fransson, "Estimating forest biomass and height using optical stereo satellite data and a dtm from laser scanning data," *Canadian Journal of Remote Sensing*, vol. 39, no. 3, pp. 251–262, 2013.
[22] Xiran Zhou, Wenwen Li, and Samantha T. Arundel, "A spatio-contextual probabilistic model for extracting linear features in hilly terrains from high-resolution dem data," *International Journal of Geographical Information Science*, vol. 33, no. 4, pp. 666–686, 2019.
[23] The Apache Software Foundation, "Apache spark - unified analytics engine for big data," `http://spark.apache.org/`, Accessed on 01/22/2019.
[24] Universidade de Santiago de Compostela (USC), "Laboratorio do territorio (laborate)," `http://laborate.usc.es/index.html`, Accessed on 01/22/2019.
[25] Infraestructura de Datos Espaciales de España (IDEE), "Geoportal idee," `http://www.idee.es/`, Accessed on 02/20/2019.
[26] University of California (San Diego), "Opentopography," `http://www.opentopography.org/`, This material is based on LiDAR Point Cloud Data Distribution and Processing services provided by the OpenTopography Facility with support from the National Science Foundation under NSF Award Numbers 1226353 & 1225810. Accessed on 02/20/2019.

Una única arquitectura neuronal profunda para eliminar ruido en imágenes hiperespectrales

Alessandro Maffei[1], Mercedes E. Paoletti[2], Juan M. Haut[2], Javier Plaza[2], Lorenzo Bruzone[1] y Antonio Plaza[2]

Resumen— **Las imágenes hiperespectrales se utilizan ampliamente en aplicaciones de observación remota de la tierra. La eliminación de ruido es un paso de preprocesamiento común antes de aplicar otras tareas de análisis e interpretación de este tipo de imágenes, como la clasificación, demezclado y la detección de objetivos. En este trabajo, desarrollamos un nuevo método espectral-espacial que elimina ruido en imágenes hiperespectrales a través de una red neuronal convolucional (CNN). Nuestro método, llamado *hyperspectral imaging single denoising* CNN (HSI-SDeCNN), considera las imágenes como cubos de datos 3D, y realiza el proceso de eliminación de ruido utilizando una sola red CNN, sin necesidad de re-entrenar el modelo para diferentes niveles de ruido. Los resultados experimentales, obtenidos con datos sintéticos y reales, demuestran que nuestro HSI-SDeCNN ofrece mejores resultados que otros métodos de eliminación de ruido.**

Palabras clave— **Imágenes hiperespectrales, eliminación de ruido, redes convolucionales, Graphics processing units (GPUs).**

I. Introducción

LAS imágenes hiperspectrales o *hyperspectral images* (HSI) se caracterizan por tener cientos de bandas espectrales, adquiridas en todo el espectro electromagnético [1], [2]. Este tipo de datos, adquiridos por espectrómetros de imagen para la observación remota de la tierra, se pueden ver como un cubo de datos de muy alta resolución espectral pero con baja resolución espacial. Además, estas imágenes suelen tener una importante cantidad de ruido.

Las misiones más recientes de observación remota de la tierra adquieren enormes cantidades de HSIs [3], lo cual fomenta su uso en una amplia gama de aplicaciones [4], como puede ser clasificación [5], demezclado espectral [6] y detección de objetivos [7], entre muchos otros.

El proceso de captación de estas imágenes es muy complejo, lo que provoca que los datos obtenidos posean ruido que genera variabilidad y similitud espectral entre clases [8]. Para evitar esto, a menudo se lleva a cabo una tarea previa que consiste en eliminar parte del ruido presente en las imágenes, permitiendo así una mejor interpretación de las mismas.

Es importante remarcar que las técnicas de *DeepLearning*, y en particular las *CNN*, involucran una gran cantidad de operaciones matriciales y vectoriales. La mayoría de estas operaciones se pueden paralelizar de forma fácil y masiva utilizando GPU, debido al diseño inherente con cientos/miles de *cores* que pueden computar una o varias operaciones matriciales en paralelo, lo que comparado con una CPU con pocos (aunque rápidos) núcleos, produce una importante disminución en tiempo de cálculo.

En este trabajo, presentamos una red convolucional (CNN) [9] mejorada para realizar la eliminación de ruido en datos HSI de manera eficiente y flexible en GPUs. Esta arquitectura se denomina HSI-SDeCNN. El modelo propuesto, en lugar de centrarse solamente en la correlación espacial (como los modelos CNN 1D y 2D), aprovecha también la correlación espectral entre las bandas adyacentes presentes en la imagen. En concreto, el método propuesto realiza la eliminación de ruido de una banda a la vez, tomando como entrada cubos de datos en lugar de una banda de manera individual (como hacen otros métodos). De esta forma, conseguimos explotar la alta correlación entre bandas adyacentes para recuperar la información que se corrompe durante el proceso de adquisición de la imagen. En términos generales, la red propuesta toma como entrada $K+1$ bandas (siendo K el número de bandas adyacentes) y devuelve la banda central sin ruido.

Las principales ventajas del nuevo método HSI-SDeCNN con respecto a otros métodos de eliminación de ruido son los siguientes:

- Nuestro método proporciona una solución rápida al problema de eliminación de ruido, explotando las relaciones existentes en los datos en una dimensión espacial menor que los datos de entrada, lo que permite que la red funcione de manera muy rápida y además sin perder apenas eficacia en términos de eliminación de ruido.
- El método toma como entrada un mapa de nivel de ruido, es decir, una estimación del nivel de ruido presente en la HSI, lo que nos permite controlar la relación entre la potencia de eliminación de ruido y la conservación de los detalles en la imagen. Esto hace que nuestra red sea flexible y adaptable a múltiples niveles de ruido, sin necesidad de entrenar diferentes modelos para cada nivel.
- Nuestro método ofrece prestaciones superiores a otros métodos de eliminación de ruido, tanto en imágenes sintéticas como reales, lo que demuestra su potencial en aplicaciones prácticas.

[1]Remote Sensing Laboratory, Department of Information Engineering and Computer Science, University of Trento, 38123 Trento, Italy (e-mail: alessandro.maffei@studenti.unitn.it; lorenzo.bruzzone@unitn.it).

[2]Hyperspectral Computing Laboratory, Department of Technology of Computers and Communications, Escuela Politécnica, University of Extremadura, 10003 Cáceres, Spain (e-mail: mpaoletti@unex.es; juanmariohaut@unex.es; jplaza@unex.es; aplaza@unex.es)

II. Metodología

En esta sección presentamos el modelo HSI-SDeCNN. Los datos hiperespectrales se pueden procesar como modelos 1D (espectral), 2D (espacial) o 3D (espectral-espacial) [10]. Dado que tanto la información espacial como la espectral son útiles en el proceso de eliminación de ruido, consideramos la imagen como una estructura de datos 3D, es decir, un modelo espectral-espacial. En particular, cada HSI puede verse como un volumen de tamaño $h \times w \times B$, siendo h y w las dimensiones espaciales de la imagen y B la espectral.

La estrategia adoptada realiza la eliminación de ruido de una banda a la vez. Para este propósito, utilizamos una red que recibe como entrada un volumen de tamaño $h \times w \times K+1$, donde K es el número de bandas adyacentes con respecto a la central (es decir, la banda objetivo). La estructura general de la red propuesta se puede dividir en cuatro operaciones principales: i) submuestreo o *downsampling*, ii) concatenación de mapas de nivel de ruido, iii) mapeo no lineal, y iv) muestreo ascendente.

- La primera operación se realiza mediante una capa de *downsampling* que cambia el volumen de entrada (de tamaño $h \times w \times K+1$) en un volumen de menor dimensión espacial. El factor de escala se ha establecido en 2. En términos generales, esta operación reduce la escala de cada banda en 4 subcubos, cada uno con tamaño $w/2 \times h/2$, Este proceso se aplica a todos los canales espectrales, y los subcubos obtenidos se concatenan a lo largo de la dimensión espectral. Esta operación permite que la red sea más rápida, sin perder rendimiento.
- La segunda capa de la red concatena un mapa de nivel de ruido M a los subcubos generados, obteniendo un volumen de tamaño $h/2 \times w/2 \times 4(K+1)+1$. El mapa de nivel de ruido proporciona una estimación del nivel de ruido σ presente en la imagen. Se inserta como un mapa uniforme (todos los elementos iguales a σ) con la misma dimensión espacial que los subcubos, para evitar cualquier desajuste en términos de dimensionalidad [11]. El mapa de nivel de ruido se inserta en la CNN como información previa, lo que permite que la red controle el balance entre el rendimiento (en cuanto a eliminación de ruido) y la conservación de detalles en la imagen. Esto se debe a que, a diferencia de los métodos de aprendizaje residuales utilizados en este caso, se agrega un mapa de nivel de ruido que hace que los parámetros del modelo sean invariantes al nivel de ruido de la imagen de entrada. Esta es una de las características principales del método propuesto: permite a la red manejar diferentes niveles de ruido, así como el ruido variante espacialmente, con un solo modelo, cambiando solamente el mapa de nivel de ruido de entrada M.
- En este punto, el volumen obtenido se pasa como entrada a una CNN estándar para sea capaz de recuperar la imagen sin ruido a partir del cubo de datos ruidoso de entrada.
- La capa final de nuestro método HSI-SDeCNN es un kernel de muestreo ascendente que toma como entrada los cuatro subcubos sin ruido (de la CNN) y genera una única banda limpia. En este sentido, realiza una operación inversa o *upsampling* con respecto a la capa de *downsampling*.

Las razones principales por las que el método propuesto exhibe un mejor rendimiento es que una red estándar son dos. En primer lugar, nuestro método toma como entrada un número significativamente mayor de bandas, lo que permite a la red explotar la alta correlación espectral entre canales adyacentes, que aplican convoluciones 3D en lugar de 2D (espaciales). En segundo lugar, dado que nuestra red considera bandas superpuestas, puede aprender a partir de una mayor cantidad de datos, lo que resulta en una mayor efectividad en el proceso de reducción de ruido mucho mayor. De hecho, nuestro HSI-SDeCNN muestra un gran rendimiento (tanto en términos de reducción de ruido como de tiempo de cálculo) en comparación con otros métodos. Esto se debe principalmente a que la capa de *downsampling* permite que la red sea más rápida (sin degradar el rendimiento) y también a la utilización del mapa de nivel de ruido de entrada, que permite que la CNN logre un mejor rendimiento sin necesidad de entrenar diferentes modelos para cada nivel de ruido.

III. Resultados Experimentales y Discusión

Para evaluar nuestro nuevo método propuesto hemos utilizado imágenes sintéticas y reales. En primer lugar, para la evaluación de la efectividad del método, se han utilizado *datasets* simulados, en los que el ruido es controlado. Además, hemos utilizado también imágenes reales ruidosas. En particular, el método HSI-SDeCNN se ha comparado varios enfoques convencionales que se adoptan típicamente para eliminar ruido en HSIs: *hybrid spatial-spectral noise reduction* (HSSNR) [12], *low-rank tensor approximation* (LRTA) [13], *block matching and 4-D algorithm* (BM4D) [14], *low-rank matrix recovery* (LRMR) [15], y *MH-prediction* [16].

Para el entrenamiento del modelo propuesto, hemos seleccionado una parte de la imagen *Washington DC Mall* sin ruido (ver Fig. 1), obtenida por el sensor *Hyperspectral Digital Imagery Collection Experiment* (HYDICE). Esta imagen se ha dividido en dos partes: una para entrenar la red y la otra para evaluarla. En particular, se han realizado experimentos con los siguientes conjuntos de datos:

- *Washington DC Mall.* Se ha empleado una parte recortada de toda la imagen para los experimentos simulados, en los que se agrega *additive white Gaussian noise* (AWGN) a la imagen original.
- *Indian Pines.* Esta imagen fue adquirida por el sensor *Airborne Visible Infra-Red Imaging Spec-*

TABLA I
EVALUACIÓN CUANTITATIVA DEL MÉTODO PROPUESTO EN COMPARACIÓN CON OTROS MÉTODOS (DATOS SIMULADOS).

Noise Level	Index	HSSNR	LRTA	BM4D	LRMR	HSID-CNN	HSI-SDeCNN
$\sigma_n = 25$	MPSNR	28,018 ± 0,0024	30,672 ± 0,0033	31,136 ± 0,0025	33,029 ± 0,0023	33,050 ± 0,0028	**33,444 ± 0,0080**
	MSSIM	0,9361 ± 0,0001	0,9629 ± 0,0002	0,9685 ± 0,0002	0,9809 ± 0,0001	0,9813 ± 0,0001	**0,9822 ± 0,0000**
	MSA	8,1332 ± 0,0034	5,7962 ± 0,0056	5,0514 ± 0,0048	4,6097 ± 0,0028	4,2641 ± 0,0026	**3,9129 ± 0,0080**
$\sigma_n = 100$	MPSNR	16,314 ± 0,0065	23,175 ± 0,0048	22,577 ± 0,0054	24,310 ± 0,0047	25,296 ± 0,0043	**25,753 ± 0,0121**
	MSSIM	0,6049 ± 0,0001	0,8494 ± 0,0003	0,8119 ± 0,0002	0,8799 ± 0,0002	0,9014 ± 0,0001	**0,9121 ± 0,0002**
	MSA	24,732 ± 0,0065	9,1219 ± 0,0072	9,7611 ± 0,0068	10,468 ± 0,0074	8,4061 ± 0,0063	**7,3951 ± 0,0182**

trometer (AVIRIS) de NASA/JPL, y se ha empleado para probar la efectividad del método sobre un escenario real.

A. Experimentos con datos simulados

En esta sección presentamos cómo se llevaron a cabo los experimentos simulados y los resultados obtenidos en el conjunto de datos *Washington DC Mall*. Para realizar las pruebas, se agregó ruido AWGN a la imagen original mediante la siguiente estrategia: consideramos el mismo nivel máximo de ruido para cada banda, donde $\sigma_n = [5, 100]$. Aquí, $n \in [1, B]$ indica una banda genérica. Con el fin de evaluar el rendimiento desde un punto de vista cuantitativo, se adoptan las siguientes métricas comúnmente empleadas: MPSNR (*Mean Peak Signal-to-Noise Ratio*), MSSIM (*Mean Structural Similarity*), y MSA (*Mean Spectral Angle*).

En los experimentos simulados hemos establecido el mapa de nivel de ruido de entrada M en el mismo nivel del ruido agregado a la imagen (esto lo denominamos ruido *ground-truth*). Los rendimientos más bajos se obtienen cuando el mapa de nivel de ruido dado en la entrada a la red difiere del nivel de ruido real presente en la imagen. En términos generales, cuando configuramos el nivel de ruido de entrada para que sea más alto que el ruido *ground-truth*, esto lleva a suavizar algunos detalles de la imagen. Por otro lado, si el nivel de ruido de entrada es más bajo que del *ground-truth*, se realiza una menor eliminación, dejando algo de ruido residual en la imagen de salida. Por lo tanto, una configuración correcta del mapa de nivel de ruido es fundamental para obtener un buen rendimiento. Sin embargo, el uso de un mapa de nivel de ruido ligeramente diferente del ruido real no introduce una degradación significativa del rendimiento, lo cual indica que la red es flexible en la selección de este parámetro. Es importante destacar que todos los resultados obtenidos con HSI-SDeCNN se han obtenido con un solo modelo, entrenado con diferentes niveles de ruido de 0 a 100. La Tabla I muestra los resultados obtenidos por nuestro método y otros competidores en los experimentos simulados para niveles de ruido 25 y 100. Puede apreciarse que nuestro método supera a todos los demás, utilizando además un único modelo.

En la Fig. 2 mostramos los resultados obtenidos en una región (recortada y ampliada) de la imagen Washington DC Mall, utilizando $\sigma_n = 100$. Podemos ver que HSID-CNN y el método propuesto obtienen mejores resultados que el resto de métodos. En particular, las imágenes sin ruido proporcionadas por HSSNR y LRMR presentan ruido residual elevado, mientras que las imágenes producidas por BM4D y LRTA contienen distorsiones. En su lugar, HSID-CNN y HSI-SDeCNN generan imágenes sin ruido que son muy similares al *ground-truth*.

B. Experimentos con datos reales

En esta sección, discutimos los experimentos realizados con la imagen AVIRIS *Indian Pines*. Para verificar la efectividad del método propuesto, se ha evaluado el rendimiento del clasificador *Support Vector Machine* (SVM) con *kernel* lineal a la hora de clasificar la imagen antes y después del proceso de eliminación de ruido. Las métricas adoptadas son la *Overall Accuracy* (OA) en la clasificación, y el coeficiente *kappa*. Para entrenar al clasificador, hemos seleccionado aleatoriamente el 10 % de las muestras etiquetadas de clase, generando por tanto un conjunto desbalanceado ya que el resto de muestras (90 %) se utiliza para la evaluación.

Dado que el nivel de ruido es desconocido en imágenes reales, el algoritmo de eliminación de ruido propuesto se aplicó empíricamente, selecccionando el mapa de nivel de ruido de entrada como el que mejor rendimiento muestra entre los siguientes niveles de ruido: $\sigma = 5, 25, 50, 75, 100$. En particular, los mejores resultados en esta imagen se logran al establecer un mapa de nivel de ruido con $\sigma = 50$. Los resultados obtenidos con este nivel de ruido se muestran mostrados en la Tabla II, donde se observa la gran mejora obtenida por el método propuesto, que consigue incrementar el OA de 75,96 % a 95,58 %.

TABLA II
RESULTADOS DE CLASIFICACIÓN OBTENIDOS POR EL MÉTODO SVM ANTES (IMAGEN ORIGINAL AVIRIS INDIAN PINES) Y DESPUÉS DE REDUCIR EL RUIDO (CON NUESTRO MÉTODO HSI-SDECNN).

	Original	HSI-SDeCNN
OA	75,96 %	95,58 %
Kappa	0,7220	0,9497

C. Tiempo de ejecución

Para evaluar la eficiencia computacional del algoritmo de eliminación de ruido propuesto, comparamos los tiempos de ejecución del método propuesto con respecto al método HSID-CNN. Esto se debe a que HSID-CNN obtiene los mejores resultados en términos de tiempo de cómputo entre los métodos comparados [17]. El tiempo de ejecución se ha

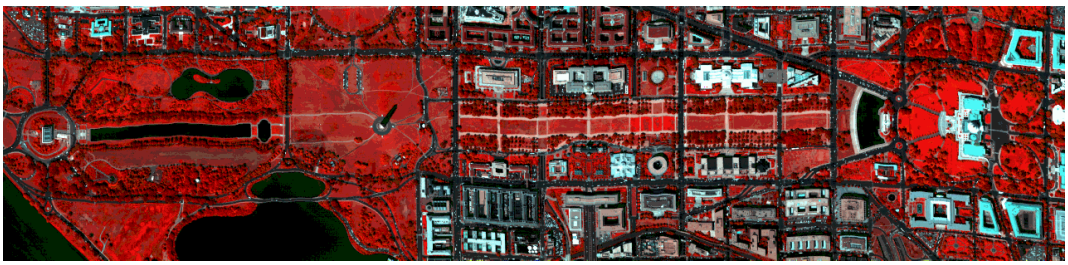
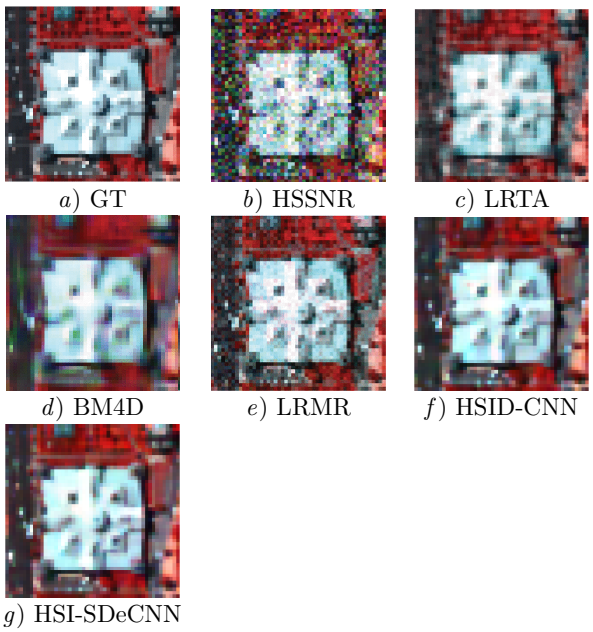


Fig. 1
Composición en falso RGB de la imagen Washington DC Mall utlizada para el entrenamiento del modelo.



a) GT *b*) HSSNR *c*) LRTA
d) BM4D *e*) LRMR *f*) HSID-CNN
g) HSI-SDeCNN

Fig. 2
Comparación de resultados obtenidos en una zona de la imagen Washington DC Mall utilizando $\sigma_n = 100$.

calculado realizando 10 ejecuciones sobre la imagen *Washington DC Mall*. Los tiempos se muestran en la Tabla III, en la que podemos observar que nuestro HSI-SDeCNN es más de dos veces más rápido que HSID-CNN.

TABLA III
Tiempo de ejecución promedio (en segundos) de HSID-CNN y HSI-SDeCNN

Dataset	Tamaño	HSID-CNN	HSI-SDeCNN
DC Mall	$200 \times 200 \times 191$	$7{,}2255 \pm 0{,}0161$	$3{,}0754 \pm 0{,}0238$

IV. Conclusiones

En este trabajo hemos presentamos un nuevo método basado en ténicas de aprendizaje profundo para la eliminación de ruido en imágenes hiperespectrales. Este método, denominado HSI-SDeCNN, toma como entrada un cubo de datos hiperespectral completo en lugar de una sola banda, lo que obliga a la red a considerar la correlación espacial y espectral presente en los datos. Las dos características principales de la red propuesta son: la capa de *downsampling*, que permite que la red sea más rápida sin perder precisión en términos de eliminación de ruido, y el mapa de nivel de ruido, que se utiliza para proporcionar a la red una estimación del ruido sin necesidad de re-entrenar el modelo. Nuestro método supera a otros métodos para la eliminación de ruido.

Agradecimientos

Este trabajo ha sido financiado por el Ministerio de Educación (Resolución de 26 de diciembre de 2014 y de 19 de noviembre de 2015, de la Secretaría de Estado de Educación, Formación Profesional y Universidades, por la que se convocan ayudas para la formación de profesorado universitario, de los subprogramas de Formación y de Movilidad incluidos en el Programa Estatal de Promoción del Talento y su Empleabilidad, en el marco del Plan Estatal de Investigación Científica y Técnica y de Innovación 2013-2016). Este trabajo también ha sido financiado por la Junta de Extremadura (Decreto 14/2018, de 6 de febrero, por el que se establecen las bases reguladoras de las ayudas para la realización de actividades de investigación y desarrollo tecnológico, de divulgación y de transferencia de conocimiento por los Grupos de Investigación de Extremadura, Ref. GR18060). El trabajo también ha sido parcialmente financiado por el programa Horizonte 2020 de la Unión Europea, Research and Innovation Programme, Grant No. 734541 (EOXPOSURE).

Referencias

[1] R. P. Iyer, A. Raveendran, S. K. T. Bhuvana, and R. Kavitha, "Hyperspectral image analysis techniques on remote sensing," in *2017 Third International Conference on Sensing, Signal Processing and Security (ICSSS)*, May 2017, pp. 392–396.

[2] Telmo Adão, Jonáš Hruška, Luís Pádua, José Bessa, Emanuel Peres, R Morais, and Joaquim Sousa, "Hyperspectral imaging: A review on uav-based sensors, data processing and applications for agriculture and forestry," *Remote Sensing*, vol. 2017, pp. 1110, 10 2017.

[3] Julie Transon, Raphaël d'Andrimont, Alexandre Maugnard, and Pierre Defourny, "Survey of hyperspectral earth observation applications from space in the sentinel-2 context," *Remote Sensing*, vol. 10, pp. 157, 01 2018.

[4] Jose Bioucas-Dias, Antonio Plaza, Gustau Camps-Valls, Paul Scheunders, N.M. Nasrabadi, and Jocelyn Chanussot, "Hyperspectral remote sensing data analysis and future challenges," *Geoscience and Remote Sensing Magazine, IEEE*, vol. 1, pp. 6–36, 06 2013.

[5] M. E. Paoletti, J. M. Haut, R. Fernandez-Beltran, J. Plaza, A. Plaza, J. Li, and F. Pla, "Capsule networks for hyperspectral image classification," *IEEE Transactions*

on Geoscience and Remote Sensing, vol. 57, no. 4, pp. 2145–2160, April 2019.

[6] R. Fernandez-Beltran, A. Plaza, J. Plaza, and F. Pla, "Hyperspectral unmixing based on dual-depth sparse probabilistic latent semantic analysis," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 56, no. 11, pp. 6344–6360, Nov 2018.

[7] B. Yang, M. Yang, A. Plaza, L. Gao, and B. Zhang, "Dual-mode fpga implementation of target and anomaly detection algorithms for real-time hyperspectral imaging," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 8, no. 6, pp. 2950–2961, June 2015.

[8] P. Ghamisi, N. Yokoya, J. Li, W. Liao, S. Liu, J. Plaza, B. Rasti, and A. Plaza, "Advances in hyperspectral image and signal processing: A comprehensive overview of the state of the art," *IEEE Geoscience and Remote Sensing Magazine*, vol. 5, no. 4, pp. 37–78, Dec 2017.

[9] ME Paoletti, JM Haut, J Plaza, and A Plaza, "A new deep convolutional neural network for fast hyperspectral image classification," *ISPRS journal of photogrammetry and remote sensing*, vol. 145, pp. 120–147, 2018.

[10] Behnood Rasti, Paul Scheunders, Pedram Ghamisi, Giorgio Licciardi, and Jocelyn Chanussot, "Noise reduction in hyperspectral imagery: Overview and application," *Remote Sensing*, vol. 10, pp. 482, 03 2018.

[11] K. Zhang, W. Zuo, and L. Zhang, "Ffdnet: Toward a fast and flexible solution for cnn-based image denoising," *IEEE Transactions on Image Processing*, vol. 27, no. 9, pp. 4608–4622, Sep. 2018.

[12] H. Othman and, "Noise reduction of hyperspectral imagery using hybrid spatial-spectral derivative-domain wavelet shrinkage," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 44, no. 2, pp. 397–408, Feb 2006.

[13] Chang Li, Yong Ma, Jun Huang, Xiaoguang Mei, and Jiayi Ma, "Hyperspectral image denoising using the robust low-rank tensor recovery," *Journal of the Optical Society of America A*, vol. 32, 09 2015.

[14] Cheng Jiang, Hongyan Zhang, Liangpei Zhang, Huanfeng Shen, and Qiangqiang Yuan, "Hyperspectral image denoising with a combined spatial and spectral weighted hyperspectral total variation model," *Canadian Journal of Remote Sensing*, vol. 42, no. 1, pp. 53–72, 2016.

[15] H. Zhang, W. He, L. Zhang, H. Shen, and Q. Yuan, "Hyperspectral image restoration using low-rank matrix recovery," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 52, no. 8, pp. 4729–4743, Aug 2014.

[16] C. Chen, W. Li, E. W. Tramel, M. Cui, S. Prasad, and J. E. Fowler, "Spectral–spatial preprocessing using multihypothesis prediction for noise-robust hyperspectral image classification," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 7, no. 4, pp. 1047–1059, April 2014.

[17] Q. Yuan, Q. Zhang, J. Li, H. Shen, and L. Zhang, "Hyperspectral image denoising employing a spatial-spectral deep residual convolutional neural network," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 57, no. 2, pp. 1205–1218, Feb 2019.

[18] K. . Dabov, A. . Foi, V. . Katkovnik, and K. . Egiazarian, "Image denoising by sparse 3-d transform-domain collaborative filtering," *Trans. Img. Proc.*, vol. 16, no. 8, pp. 2080–2095, Aug. 2007.

[19] Shuhang Gu, Lei Zhang, Wangmeng Zuo, and Xiangchu Feng, "Weighted nuclear norm minimization with application to image denoising," in *Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition*, Washington, DC, USA, 2014, CVPR '14, pp. 2862–2869, IEEE Computer Society.

[20] Yushi Chen, Zhouhan Lin, Xing Zhao, Gang Wang, and Yanfeng Gu, "Deep learning-based classification of hyperspectral data," *IEEE Journal of Selected topics in applied earth observations and remote sensing*, vol. 7, no. 6, pp. 2094–2107, 2014.

[21] Yushi Chen, Xing Zhao, and Xiuping Jia, "Spectral–spatial classification of hyperspectral data based on deep belief network," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 8, no. 6, pp. 2381–2392, 2015.

[22] Pedram Ghamisi, Yushi Chen, and Xiao Xiang Zhu, "A self-improving convolution neural network for the classification of hyperspectral data," *IEEE Geoscience and Remote Sensing Letters*, vol. 13, no. 10, pp. 1537–1541, 2016.

[23] Jürgen Schmidhuber, "Deep learning in neural networks: An overview," *Neural networks*, vol. 61, pp. 85–117, 2015.

[24] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[25] Evgeny A Smirnov, Denis M Timoshenko, and Serge N Andrianov, "Comparison of regularization methods for imagenet classification with deep convolutional neural networks," *Aasri Procedia*, vol. 6, pp. 89–94, 2014.

[26] Andrew G Howard, "Some improvements on deep convolutional neural network based image classification," *arXiv preprint arXiv:1312.5402*, 2013.

[27] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436, 2015.

[28] Yushi Chen, Hanlu Jiang, Chunyang Li, Xiuping Jia, and Pedram Ghamisi, "Deep feature extraction and classification of hyperspectral images based on convolutional neural networks," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 54, no. 10, pp. 6232–6251, 2016.

[29] Lichao Mou, Pedram Ghamisi, and Xiao Xiang Zhu, "Deep recurrent neural networks for hyperspectral image classification," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 55, no. 7, pp. 3639–3655, 2017.

[30] Ying Li, Haokui Zhang, Xizhe Xue, Yenan Jiang, and Qiang Shen, "Deep learning for remote sensing image classification: A survey," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 8, no. 6, pp. e1264, 2018.

[31] Xiaofei Yang, Yunming Ye, Xutao Li, Raymond YK Lau, Xiaofeng Zhang, and Xiaohui Huang, "Hyperspectral image classification with deep learning models," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 56, no. 9, pp. 5408–5423, 2018.

[32] Burkni Palsson, Jakob Sigurdsson, Johannes R Sveinsson, and Magnus O Ulfarsson, "Hyperspectral unmixing using a neural network autoencoder," *IEEE Access*, vol. 6, pp. 25646–25656, 2018.

[33] K. Zhang, W. Zuo, Y. Chen, D. Meng, and L. Zhang, "Beyond a gaussian denoiser: Residual learning of deep cnn for image denoising," *IEEE Transactions on Image Processing*, vol. 26, no. 7, pp. 3142–3155, July 2017.

[34] Mukesh C Motwani, Mukesh C Gadiya, Rakhi C Motwani, and Frederick C Harris, "Survey of image denoising techniques," in *Proceedings of GSPX*, 2004, vol. 2004.

[35] Qiangqiang Yuan, Liangpei Zhang, and Huanfeng Shen, "Hyperspectral image denoising employing a spectral-spatial adaptive total variation model," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 50, pp. 3660–3677, 10 2012.

[36] Matteo Maggioni, Vladimir Katkovnik, Karen Egiazarian, and Alessandro Foi, "Nonlocal transform-domain filter for volumetric data denoising and reconstruction," *Trans. Img. Proc.*, vol. 22, no. 1, pp. 119–133, Jan. 2013.

[37] T. Lu, S. Li, L. Fang, Y. Ma, and J. A. Benediktsson, "Spectral–spatial adaptive sparse representation for hyperspectral image denoising," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 54, no. 1, pp. 373–385, Jan 2016.

[38] Damien Letexier and Salah Bourennane, "Noise removal from hyperspectral images by multidimensional filtering," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 46, pp. 2061–2069, 2008.

[39] Y. Zhao and J. Yang, "Hyperspectral image denoising via sparse representation and low-rank constraint," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 53, no. 1, pp. 296–308, Jan 2015.

[40] W. He, H. Zhang, H. Shen, and L. Zhang, "Hyperspectral image denoising using local low-rank matrix recovery and global spatial–spectral total variation," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 11, no. 3, pp. 713–729, March 2018.

[41] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: from error visibility to structural similarity," *IEEE Transactions on Image Processing*, vol. 13, no. 4, pp. 600–612, April 2004.

[42] J. D. O'Sullivan, P. R. Hoy, and H. N. Rutt, "An extended spectral angle map for hyperspectral and multispectral imaging," in *CLEO: 2011 - Laser Science to Photonic Applications*, May 2011, pp. 1–2.

[43] Jie Li, Qiangqiang Yuan, Huanfeng Shen, and Liangpei Zhang, "Hyperspectral image recovery employing a mul-

tidimensional nonlocal total variation model," *Signal Processing*, vol. 111, pp. 230 – 248, 2015.

[44] Diederik P. Kingma and Jimmy Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2014.

[45] Andrea Vedaldi and Karel Lenc, "Matconvnet - convolutional neural networks for MATLAB," *CoRR*, vol. abs/1412.4564, 2014.

[46] M. Ye and Y. Qian, "Mixed poisson-gaussian noise model based sparse denoising for hyperspectral imagery," in *2012 4th Workshop on Hyperspectral Image and Signal Processing: Evolution in Remote Sensing (WHISPERS)*, June 2012, pp. 1–4.

[47] D. Manolakis, R. Lockwood, and T. Cooley, "On the spectral correlation structure of hyperspectral imaging data," in *IGARSS 2008 - 2008 IEEE International Geoscience and Remote Sensing Symposium*, July 2008, vol. 2, pp. II–581–II–584.

[48] Matias Tassano, Julie Delon, and Thomas Veit, "An analysis and implementation of the ffdnet image denoising method," *Image Processing On Line*, vol. 9, pp. 1–25, 01 2019.

[49] Michaël Gharbi, Gaurav Chaurasia, Sylvain Paris, and Frédo Durand, "Deep joint demosaicking and denoising," *ACM Transactions on Graphics*, vol. 35, pp. 1–12, 11 2016.

[50] D. Landgrebe, "Hyperspectral image data analysis," *IEEE Signal Processing Magazine*, vol. 19, no. 1, pp. 17–28, Jan 2002.

[51] Matthew D. Zeiler and Rob Fergus, "Visualizing and understanding convolutional networks," *CoRR*, vol. abs/1311.2901, 2013.

[52] I. Dopido, J. Li, A. Plaza, and J. M. Bioucas-Dias, "A new semi-supervised approach for hyperspectral image classification with different active learning strategies," in *2012 4th Workshop on Hyperspectral Image and Signal Processing: Evolution in Remote Sensing (WHISPERS)*, June 2012, pp. 1–4.

[53] Maider Vidal and José Amigo, "Pre-processing of hyperspectral images. essential steps before image analysis," *Chemometrics and Intelligent Laboratory Systems*, vol. 117, pp. 138–148, 08 2012.

Framework escalable para monitorización y planificación de aplicaciones paralelas

Alberto Cascajo, David E. Singh y Jesus Carretero [1]

Resumen—
En este trabajo presentamos un entorno de ejecución HPC que proporciona nuevas estrategias para la gestión de la plataforma y la planificación de tareas paralelas. Este trabajo incluye una herramienta de monitorización que es capaz de analizar los nodos de cómputo de la plataforma y detectar riesgos de contención presentes en ellos. También se presenta un planificador que utiliza la información proporcionada por el monitor, en combinación con monitorización a nivel de aplicación, para evitar los riesgos de contención mediante la migración dinámica de aplicaciones. En el trabajo se incluye una descripción de la arquitectura así como una evaluación práctica de la propuesta, la cual muestra mejoras significativas en el rendimiento de las aplicación mediante su uso.

Palabras clave—**Aplicaciones paralelas, herramientas de monitorización, planificación, maleabilidad.**

I. Introducción

En la actualidad un desafío importante en la computación paralela a gran escala consiste en desarrollar componentes escalables que operen de manera coordinada para un uso más eficiente de la plataforma. En los últimos años, los investigadores han estudiado cómo mejorar este modelo de programación, basando sus soluciones en los estudios de perfiles de aplicaciones [1]. Tradicionalmente, en HPC, los programadores asignan nodos de cómputo de forma estática, intentando ejecutar diferentes aplicaciones en diferentes nodos para evitar interferencias. Además, el número de núcleos por nodo de cómputo aumenta cada año. Esto hace que sea demasiado costoso asignar nodos de cómputo exclusivos a las aplicaciones dado que parte de los recursos asignados podrían no ser utilizados. La alternativa a este problema es compartir los nodos de cómputo entre diferentes aplicaciones. Sin embargo, en este caso el rendimiento de ambas aplicaciones podría verse reducido por potenciales riesgos de contención existentes al acceder a los recursos compartidos.

Este trabajo aborda una solución a estos desafíos desde dos direcciones diferentes: primero, presentamos un entorno de monitorización y control que funciona de manera coordinada. En segundo lugar, presentamos una nueva estrategia de programación que permite compartir los nodos de cómputo entre diferentes aplicaciones, superando los riesgos relacionados con la contención por la compartición de recursos. El entorno de ejecución integra tres componentes diferentes: un monitor ligero y escalable [2] que analiza los nodos de cómputo y detecta riesgos de contención, una extensión de FlexMPI [3] que proporciona maleabilidad y monitorización a las aplicaciones y un planificador de tareas que combina la información del monitor a nivel del sistema con la monitorización a nivel de la aplicación (provisto por FlexMPI) para tomar decisiones de planificación teniendo en cuenta el rendimiento.

La segunda contribución es una nueva política de planificación de aplicaciones paralelas que permite ejecutar diferentes aplicaciones en el mismo nodo de cómputo. En el caso de producirse una degradación del rendimiento relacionada con la contención, dicha política incluye una metodología para detectarla y evitarla dinámicamente mediante la migración de aplicaciones utilizando la maleabilidad. Este trabajo proporciona una descripción detallada de la propuesta, así como una evaluación práctica en una plataforma real.

Por lo que sabemos, este es el primer trabajo que combina estas tres funciones (monitorización a nivel de sistema/aplicación, planificación y maleabilidad) en un entorno unificado para mejorar la eficiencia en el uso de recursos del sistema, así como tratar de reducir el consumo energético gracias la reducción del número de nodos de cómputo en uso.

II. Arquitectura del Sistema

La figura 1 muestra una descripción del entorno de ejecución. La parte central corresponde a los nodos de cómputo del clúster organizados en dos grupos (o *racks*). La mitad superior de la figura se corresponde a la infraestructura del monitor de sistema, llamado DaeMon, que gestiona los recursos de la plataforma. Consiste en un *proceso Daemon Monitor* (DM) por nodo de cómputo que recopila periódicamente diferentes métricas relacionadas con el estado del nodo. Los *procesos DaeMon Aggregators* (DA) recopilan información de los DMs (flecha 1) y los retransmite al *proceso DaeMon Server* (DS) (flecha 2), que procesa y almacena la información en una base de datos en memoria e incluye un canal de comunicación con los usuarios.

Esta infraestructura está diseñada para proporcionar escalabilidad de dos maneras distintas. Primero, la lógica del monitor se distribuye en una configuración topológica adaptable a cada entorno. Los DM, los DA y los DS se interconectan mediante un esquema basado en grafos que se puede mapear a la topología de red usada por el clúster. El segundo mecanismo de escalabilidad tiene como objetivo reducir la sobrecarga de red y memoria. Hemos incluido en cada DM un método de análisis en el nodo que reduce el volumen de datos enviado a los DA y DS al evitar la transmisión de aquellas muestras (s_i) que son similares a la última muestra enviada (s_{i-1}), donde

[1]Universidad Carlos III de Madrid.Leganés, Madrid, Spain. e-mail {acascajo,dexposit,jcarrete}@inf.uc3m.es

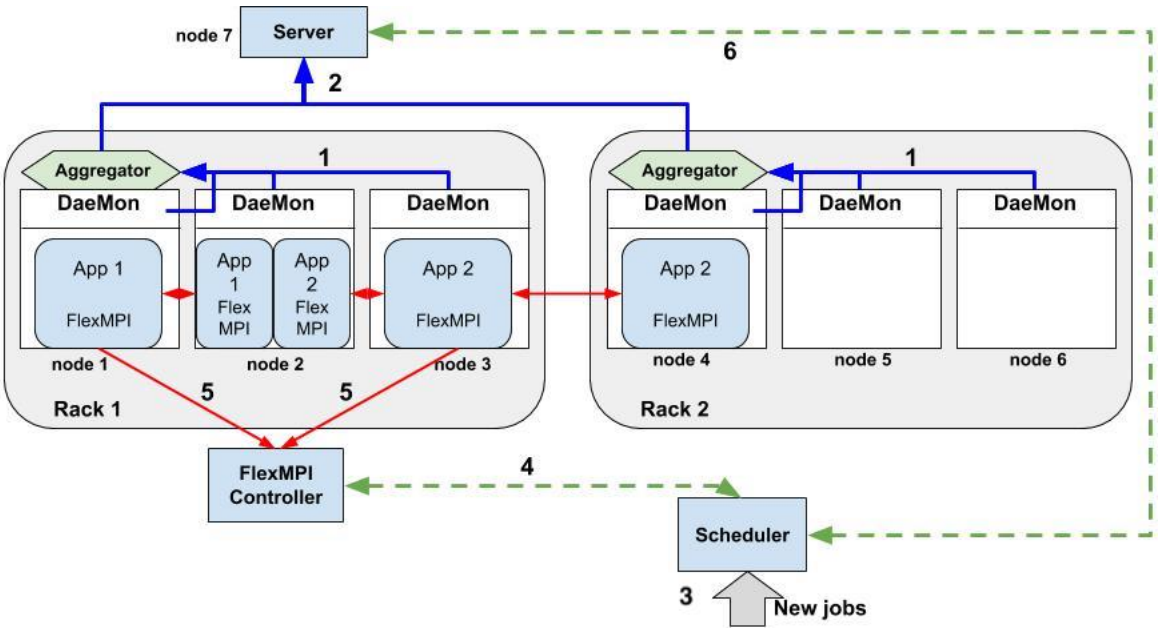


Fig. 1. Diagrama con la arquitectura del sistema.

la similitud se define utilizando una tolerancia dada $+/-tol$.

En este trabajo consideramos tres clases diferentes de contención: la memoria del nodo se agota (RAM hot-spot), los conflictos en los accesos a la interfaz de red del nodo (NET hot-spot) y conflictos relacionados con la caché (CACHE hot-spot). La contención se cuantifica como el porcentaje de uso de RAM y el ancho de banda de la red y el porcentaje de fallos de la caché último nivel.

La mitad inferior de la Figura 1 se corresponde al *Controlador de aplicaciones* que incluye el entorno de FlexMPI y el planificador del sistema que realiza el análisis y la coordinación de las aplicaciones en ejecución. FlexMPI consiste en una biblioteca ejecutada con las aplicaciones y un controlador central que recibe los comandos del programador (flecha 4) y recopila las métricas de monitorización relacionadas con la aplicación (flecha 5). El planificador es responsable de ejecutar las aplicaciones aprovechando la información proporcionada tanto por el monitor del sistema como por las aplicaciones para reducir las interferencias entre las mismas.

Hay que tener en cuenta que el diseño de este entorno es escalable dado que los DAs reciben tantas conexiones como nodos por *rack*, y están conectados con el servidor de DaeMon en forma de grafo. Por parte del controlador FlexMPI, este tiene solo una conexión por aplicación, y el resto de los componentes están interconectados siguiendo un esquema punto a punto. Todos estos componentes son aplicaciones multihilo. Por razones de espacio y simplicidad, en este documento solo describimos su implementación secuencial.

A continuación describimos el funcionamiento del entorno con un ejemplo práctico. Supongamos que cada nodo de cómputo en la Figura 1 consta de 12 núcleos y que hay dos aplicaciones con 18 y 30 procesos listos para ejecutarse. La aplicación 1 se ejecuta primero en los nodos 1 y 2 con 12 y 6 procesos, respectivamente. El programa supervisa la aplicación (utilizando FlexMPI) y extrae diferentes métricas de rendimiento. Luego, la aplicación 2 se ejecuta de manera similar, siendo asignada a los nodos de cómputo 3, 4 y 5, con 12, 12 y 6 procesos respectivamente. Tenga en cuenta que todos los nodos son exclusivos de la aplicación, por lo tanto, cuando las mediciones de rendimiento son recopiladas por FlexMPI, no existe riesgo de conflicto con otras aplicaciones. Después de esto, los 6 procesos en el nodo de cómputo 5 se migran dinámicamente (usando FlexMPI) para al nodo 2, que se convierte en un nodo compartido. Hay que tener en cuenta que con esta estrategia solo se utilizan cuatro nodos de cómputo (en lugar de cinco), lo que ahorra recursos y energía.

Supongamos ahora que las aplicaciones 1 y 2 tienen contención relacionada con la memoria caché en el nodo 2. Esta interferencia es detectada por el proceso DM ejecutado en el nodo y posteriormente el servidor alerta al planificador sobre dicho riesgo. A continuación el planificador activa selectivamente la monitorización a nivel de aplicación para las aplicaciones ejecutadas en el nodo y compara las métricas obtenidas con las originales. En el caso de detectar una degradación del rendimiento, los 6 procesos de la aplicación 2 se mueven a otro nodo libre exclusivo para esta aplicación. Tenga en cuenta que ahora el nodo 2 tiene 6 núcleos disponibles para otras posibles aplicaciones entrantes.

A. Monitor del Sistema - DaeMon

El algoritmo 1 muestra la lógica del monitor DaeMon. La primera parte se ejecuta en cada uno de los nodos donde está ejecutándose el proceso DM, que recopila todas las métricas y las envía a DA sólo si las medidas están fuera del rango de tolerancia. La segunda parte se ejecuta en el DA, que recibe eventos mediante la función *receive_event*() y redirige es-

Algorithm 1 Algoritmo distribuido de DaeMon.

```
1:  while running do
2:    // Daemon Monitor logic
3:    collect_node_metrics(e_i)
4:    send_by_tolerance(node_i, e_i)
5:    // DaeMon Aggregator logic
6:    if receive_event(node_i, e_i) then
7:      update_node(node_i, e_i)
8:    end if
9:    // DaeMon Server logic
10:   receive_event(node_i, e_i)
11:   update_db_node(node_i, e_i)
12:   status = obtain_status(node_i)
13:   if status > 0 then
14:     notify_scheduler(node_i, status)
15:   end if
16:   if allocate(app_i, Δp, excl) then
17:     if is_new_application(app_i) then
18:       nodes = allocate_new(Δp)
19:     else
20:       nodes = reallocate(app_i, Δp, excl)
21:     end if
22:     return_scheduler(nodes)
23:   end if
24: end while
```

Algorithm 2 Algoritmo de planificación.

```
1:  while running do
2:    if app_i = new_application() then
3:      put_in_queue(app_i, Q)
4:    end if
5:    if app_i = is_ready(Q), ∀app_i ∈ Q then
6:      nodes = allocate(app_i, Δp_i, 0)
7:      execute(app_i, nodes)
8:      S_i^init = FlexMPI_Monitor(app_i, 0)
9:    end if
10:   if {node_i, contention} = notify_scheduler() then
11:     if contention == RAM then
12:       app_j = latest_application_id(node_i)
13:     else if (contention == CACHE || contention ==
        NET) then
14:       S_i^curr = FlexMPI_monitor(app_i, contention)
15:       ∀app_i ∈ node_i,
16:       app_j = evaluate_contention(S^init, S^curr)
17:     end if
18:     if app_j ≠ 0 then
19:       Δp = num_processes(app_j, node)
20:       nodes = return_scheduler(app_j, −Δp, 0)
21:       FlexMPI_remove(nodes, app_j, −Δp, 0)
22:       nodes = allocate(app_j, Δp, 1)
23:       FlexMPI_spawn(nodes, app_j, Δp, 0)
24:     end if
25:   end if
26: end while
```

ta información hacia el servidor mediante la función *update_node*(). Esta última función se comunica con el Servidor DaeMon (DS), donde $node_i$ es el nodo de cálculo y e_i es el evento relacionado. Se actualiza la base de datos con la nueva medida, y posteriormente, para cada nuevo evento, *obtain_state*() devuelve el estado actual del nodo i. Los valores de estado posibles son: NINGUNO, sin disputa, y los hot-spots de RAM, NET y CACHE antes mencionados. Si alguno de los umbrales establecidos se ve sobrepasado, el servidor informa al planificador sobre el tipo de contención por medio de $notify\_scheduler()$ (flecha 6 en Figura 1).

La tercera parte del algoritmo se ejecuta en el DS. Además de la función de recepción y almacenamiento de las métricas, la función *allocate*() recibe nuevos comandos del planificador (flecha 6) para asignar nuevos recursos. Los argumentos de entrada son el ID

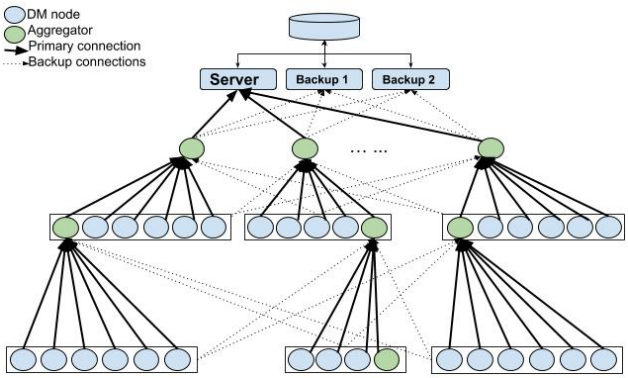


Fig. 2. Ejemplo de despliegue del monitor.

de la aplicación (app_i), el número de procesos que se deben asignar ($Deltap$) y una señal que indica si los nodos de cómputo deben ser exclusivos ($excl = 1$) o no ($excl = 0$). Usando el valor app_i es posible distinguir si la solicitud está relacionada con una nueva, que debe ejecutarse, o una aplicación en ejecución existente, que debe moverse mediante maleabilidad. Hay que tener en cuenta que la política de asignación depende de esto. Por un lado, para las nuevas aplicaciones, *allocate_new*() devuelve los nodos asignados. Notar que estos nodos pueden no ser exclusivos para la aplicación. Por otro lado, para aplicaciones existentes, *reallocate*() devuelve los nodos relacionados que son exclusivos según el valor del argumento *excl*. Tenga en cuenta que $Deltap$ puede ser negativo en caso de reducir el tamaño de la aplicación o ser positivo. Finalmente, *return_scheduler*() envía la lista de nodos seleccionados al planificador del sistema.

A.1 Despliegue del sistema

En la Figura 2 se puede ver un lanzamiento del monitor de sistema con una topología jerárquica. Conviene aclarar que este modelo es únicamente un ejemplo para mostrar las capacidades de escalabilidad, pero el usuario puede adaptar el despliegue a cualquiera topología del sistema (circular, de estrella, etc.). Notar que el Agregador (DA) recopila las métricas de los nodos que tiene asociados, y envía dicha información a la capa superior. Este envío puede realizarse a otro DA (siguiendo la topología del sistema) o al DS. Como se ha comentado, DA es el encargado de proveer al sistema de escalabilidad, y tras un estudio realizado se puede hemos establecido que cada DA es capaz de mantener asociados 200 nodos en el peor de los casos.

B. Control de las aplicaciones

El algoritmo 2 muestra el pseudocódigo del planificador, encargado de gestionar la utilización de los nodos de cómputo por parte de las aplicaciones.

Mediante *new_application*(), cuando una nueva aplicación app_i llega al sistema, es gestionada por el planificador y almacenada en una cola de espera Q (línea 3). La función $is\_ready()$ evalúa todas las aplicaciones app_i en Q y devuelve *true* cuando hay suficientes recursos para ejecutar una de ellas. En este caso, la función *allocate*() proporciona (a través de DaeMon Server, flecha 6) los nodos de proceso asig-

nados a la aplicación y $FlexMPI\_Monitor()$ instrumenta la aplicación (flechas 4 y 5) para obtener varias métricas de rendimiento de referencia (S_i^{init}), incluyendo tiempos de usuario, sistema y de comunicaciones así como otros valores relacionados con contadores de rendimiento. Esta supervisión sólo está activa durante un corto período de tiempo cuando la aplicación comienza a ejecutarse. Para obtener métricas de rendimiento no afectadas por la contención, las aplicaciones se ejecutan inicialmente en nodos exclusivos. Una vez que se obtienen estas métricas, los procesos que se ejecutan en el último nodo asignado, el que ejecuta un menor número de procesos, se migran a un nodo de cómputo compartido en el caso en que esté disponible disponible.

La segunda parte del planificador está relacionada con el análisis de contención. La función $notify\_scheduler()$ devuelve la información proporcionada por el monitor sobre aquellos nodos de cómputo ($node_i$) con posibles riesgos de contención. La clase de contención está codificada en la variable *contention*. En caso de contención de memoria principal, la aplicación conflictiva se migra inmediatamente para disminuir la memoria principal en uso. Para la contención relacionada con la caché y la red, por medio de $FlexMPI\_Monitor()$, se monitorizan todas las aplicaciones en conflicto, obteniendo, para cada una de ellas, las métricas de rendimiento actuales S_i^{curr}. La función $eval\_contention()$ determina si alguna de las aplicaciones en el nodo de cómputo tiene una degradación de rendimiento. Esto se realiza comparando las métricas existentes con las originales. Si no hay cambios en el rendimiento, la señal de contención se inhibe posteriormente durante un cierto tiempo o hasta que se ejecute una nueva aplicación en el nodo. En el otro caso, cuando existe una degradación del rendimiento superior al 10 %, se selecciona una aplicación (app_j) y se migra a un nodo exclusivo. Existen varias políticas para seleccionar la aplicación que se va a migrar. En nuestros experimentos, la política consiste en migrar la última aplicación en ejecución. La operación de migración se realiza mediante dos operaciones, primero eliminando los procesos de Δp (mediante la maleabilidad) del nodo actual (líneas 14, 15 y 16) y luego creando los procesos de Δp en el nuevo nodo proporcionado por el monitor (líneas 18 y 19).

III. Evaluación

En esta sección se describe la plataforma y las aplicaciones que se han utilizado, así como los experimentos y los resultados que se han obtenido en las distintas pruebas realizadas.

A. Entorno de pruebas

Para los experimentos hemos utilizado un grupo heterogéneo de ocho nodos divididos en dos racks. La conexión entre nodos en el mismo rack es una red Ethernet de 10 Gbps, mientras que la conexión entre los distintos racks se realiza a través de una red Ethernet de 1 Gbps. El clúster contiene dos nodos con procesador Intel (R) Xeon (R) E5 con 8 núcleos y 256 GB de RAM en un rack y seis nodos con procesador Intel (R) Xeon (R) E7 con 12 núcleos y 128 GB de RAM en el otro.

Todos los casos de uso que se han seleccionado se componen de una serie de aplicaciones reales y kernels sintéticos. Todos ellos han sido integrados con FlexMPI. Las pruebas se han realizado combinando la ejecución de distintas aplicaciones entre sí, tratando de generar casos con contención y casos sin ella. La Tabla I muestra las características de los casos de uso.

EpiGraph [4] es un simulador paralelo estocástico de propagación del virus de la gripe. Este programa está configurado para simular la propagación en la ciudad de Bilbao usando un grado no dirigido de 703,258 nodos y 8,806,520 vértices que corresponde a las conexiones a nivel de individuo del entorno simulado. *Jacobi* es el kernel de método iterativo de Jacobi que opera con matrices densas. CG es el kernel del método iterativo del Gradiente Conjugado que realiza multiplicaciones de matrices dispersas con vectores. Se han evaluado dos matrices dispersas con el fin de analizar el impacto de la localidad de datos en la interferencias. La primera de ellas, denominada B_{ll} (CG kernel with **l**ow **l**ocality), es una matriz dispersa cuadrada con 500,000 filas/columnas y 39,996,827 elementos no nulos. La segunda matriz, denominada B_{hl} (CG kernel with **h**igh **l**ocality), se corresponde con una matriz dispersa de 500,000 filas/columnas y 39,967,238 elementos no nulos. En este caso los elementos no nulos se distribuyen de forma aleatorio en una banda de tamaño de 20,000 entradas. Notar que este patrón tiene asociado una mayor localidad en el acceso a los elementos del vector.

CPU es una variante del kernel de Jacobi que no incluye comunicaciones. Representa una aplicación con exclusivamente tiempo de CPU y que se divide en dos subclases: D_m con un tamaño de memoria media que usa seis matrices densas del 20,000 entradas y D_{xl} (e**x**tra-**l**arge) que opera con seis matrices densas de 50,000 entradas (con un tamaño total de 120 GB). CPUNET es una variación del kernel D_m que alterna fases intensivas de CPU y comunicación. Este kernel se utiliza para evaluar la contención de la red. *RMEM* y *IMEM* son kernels intensivos en memoria que operan con seis matrices cuadradas de 20,000 entradas (19.2GB) de forma regular e irregular, respectivamente. Ambos kernels son usados para introducir carga en la memoria caché de los nodos.

B. Monitor DaeMon

En esta sección se presenta un análisis cuantitativo del rendimiento del monitor de sistema DaeMon. Se ha evaluado la sobrecarga que genera el monitor así como el efecto que produce la optimización para reducir el uso de red. La Tabla II muestra el impacto del monitor con dos intervalos de medición distintos (uno bajo y uno alto). Este intervalo indica el tiempo en el que el monitor recolecta las métricas y las envía. Se puede observar que en ambos casos la sobrecarga

TABLA I
CASOS DE USO CONSIDERADOS.

Código	Nombre	Clase	Patrón acceso	Tamaño	Intensivo en
A	EpiGraph	Application	irregular	small	CPU & network
B_{ll}, B_{hl}	CG	kernel	irregular	small	CPU
C	Jacobi	kernel	regular	medium	CPU
D_m	CPU	Synthetic	regular	medium	CPU
D_{xl}	CPU	Synthetic	regular	large	CPU & memory
E	CPUNET	Synthetic	regular	medium	CPU & network
F	IMEM	Synthetic	irregular	medium	memory

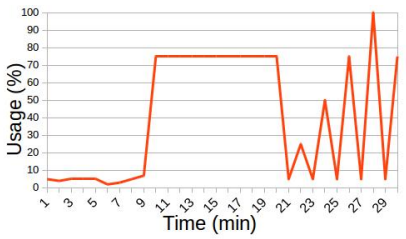


Fig. 3. CPU evolution during evaluation of the tracks.

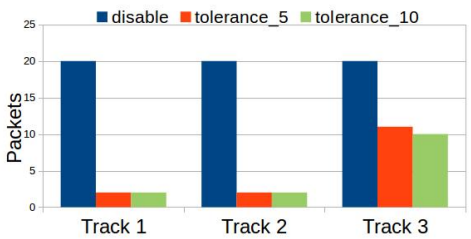


Fig. 4. Network traffic with/without in-node analysis.

en CPU y memoria son bajos en ambos casos.

Para evaluar el efecto de la optimización denominada *in-node analysis* se ha utilizado la ejecución de un *workflow* con las siguientes características de ejecución (Figura 3):

- Tres fases de 10 minutos cada una.
- Fase 1 T=[0m, 10m]: nodo sin carga de cómputo.
- Fase 2 T=[10m, 20m]: nodo con carga de cómputo constante del 75 %.
- Fase 3 T=[20m, 30m]: nodo con carga que varía abruptamente cada 30 s.

La figura 4 muestra el tráfico de red relacionado con el monitor cuando el filtrado de los eventos del nodo está habilitado (con tolerancias del 5 % y 10 %), así como cuando no lo está. Los resultados del experimento muestran que el filtrado reduce drásticamente el tráfico de la red (hasta un 90 % en las fases 1 y 2). Incluso con carga variable (fase 3), se puede reducir casi el 50 % del tráfico del monitor. En todos los casos, el servidor DaeMon obtiene el estado real del sistema dentro de los límites de tolerancia.

Adicionalmente, hemos realizado simulaciones del monitor mediante las herramienta [5] y [6], verificado que un solo DA o DS puede atender a más de 200 conexiones cuando la política de filtrado está deshabilitada. Este valor es para el peor de los casos de estudio, con un nodo DS que tiene un único hilo procesador de información y el intervalo de tiempo de medición está establecido en un segundo.

C. Planificador de aplicaciones

Primero, se evalúa un caso de uso *ad hoc* que ilustra el impacto de cada clase de interferencia, así como la sobrecarga relacionada con el entorno propuesto. Para los experimentos, el código fue compilado con gcc 7.4.0 y MPICH 3.2. Este caso de uso consiste en una secuencia de aplicaciones que se ejecutan en tres nodos de cómputo compartidos. La Tabla III muestra la estructura y los resultados para este caso de uso. El orden de ejecución (también utilizado como id de aplicación) se muestra en la primera columna de la tabla. Para cada código, la tabla incluye el nombre, el número de procesadores y la huella de memoria de cada aplicación. La etiqueta *Shared* muestra el ID de la aplicación que se ejecuta en el mismo nodo de cómputo. Por ejemplo, las aplicaciones 1 y 2 están ubicadas en el mismo nodo compartido, lo que produce un aumento de la tasa de fallos de caché de último nivel detectada por el monitor. Una vez que el programador recibe la notificación, se utiliza FlexMPI para monitorizar ambas aplicaciones. T1, T2 y T3 son el tiempo de ejecución antes de la interferencia, durante la interferencia y después de la interferencia, respectivamente. Podemos observar en la tabla que la aplicación 1 duplica el tiempo de ejecución mientras que la aplicación 2 no se ve afectada (principalmente porque no tiene localidad de datos temporales). Para evitar esta degradación del rendimiento, la aplicación 2 se mueve a un nodo exclusivo. La sobrecarga de reconfiguración se muestra en la última columna (32,2 segundos). Hay que tener en cuenta que esta sobrecarga corresponde a cuatro reconfiguraciones: (1) creando 8 procesos en el nodo compartido, (2) destruyendo 8 procesos en el nodo exclusivo inicial, y luego, después del análisis, (3) creando 8 procesos en el nuevo nodo exclusivo y (4) destruyendo 8 procesos en el nodo compartido. Para la aplicación 2, la sobrecarga está relacionada principalmente con la redistribución de datos.

En este ejemplo las aplicaciones 3 y 4 producen un conflicto en un nodo compartido que no produce ninguna degradación del rendimiento (debido a la buena ubicación de los datos para ambos), por lo que no se necesita una reconfiguración de las mismas. Las aplicaciones 5 y 6 producen interferencias de comunicación que afectan a ambas. La aplicación

TABLA II

Sobrecarga de DM por nodo, y de DS en en entorno de pruebas con *intervalos de medición* de 30 y 2 secs.

Component	Network usage (bytes)	Memory (MB)	CPU use (%)	
			30 s.	2 s.
Monitor	1472 *per period*	10	0,01	5 %
Server	1472 *per hot_spot*	4	0,01	5 %

TABLA III

Ejemplo de casos de uso de contención (tiempos en sec.).

Id.	Code	Procs.	Size (Gb)	Shared	T1	T2	T3	Overhead
1	B_{ll}	4	0,3	2	3,2	6,4	3,2	-
2	F	20	17,9	1	29,6	29,2	29,1	32,2
3	B_{hl}	4	0,3	4	2,8	2,8	2,8	-
4	F	20	17,9	3	29,5	29,6	29,5	-
5	E	20	1,2	6-7	6,9	9,0	7,0	-
6	E	20	1,2	5	7,0	8,78	7,6	2,8
7	D_m	20	1,2	5	0,1	0,1	0,1	-
8	D_{xl}	6	94,6	9	21,0	27,8	21,6	-
9	D_{xl}	16	111,8	8	9,4	9,6	9,4	101,1

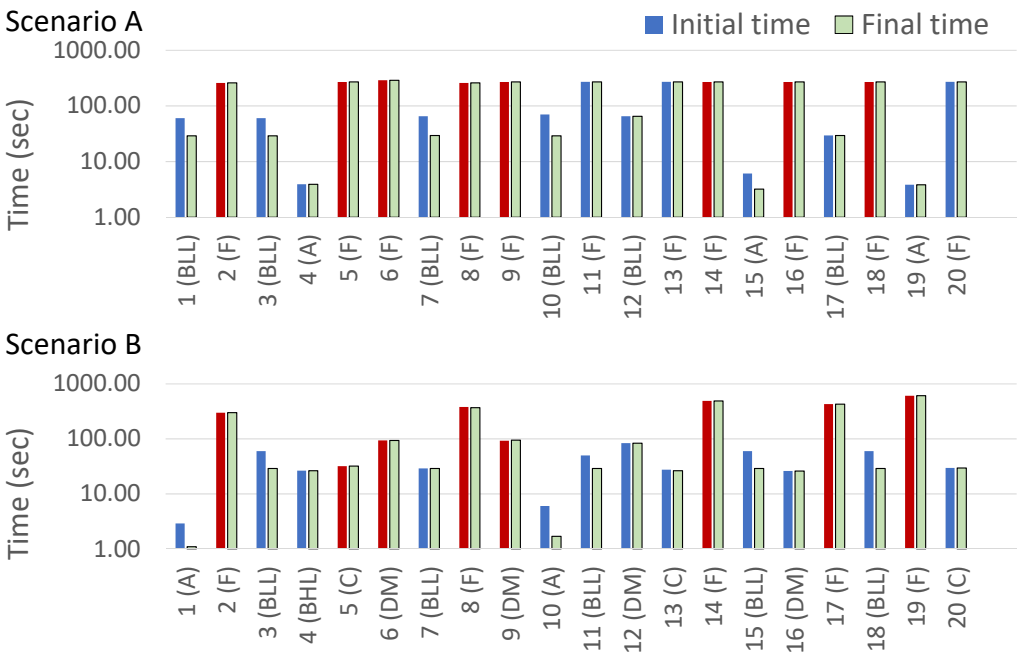


Fig. 5. Evaluación del framework para dos escenarios.

6 se migra a un nodo exclusivo que se encuentra en un rack diferente, lo que aumenta el tiempo de ejecución final (T3) en comparación con el inicial (T1) debido a un ancho de banda de red más lento. Dada la pequeña cantidad de datos, la sobrecarga está relacionada principalmente con la creación/destrucción de procesos. Después de esta migración, la aplicación 7 se coloca en el mismo nodo que 5, sin producir degradación ya que ambas aplicaciones tienen diferentes perfiles.

Finalmente, las aplicaciones 8 y 9 casi alcanzan la capacidad máxima de memoria del nodo de cómputo en el que se ejecutan. Esto lo detecta el monitor y la aplicación 9 se migra inmediatamente a un nodo libre. Hay que tener en cuenta que la sobrecarga es más importante debido a la gran cantidad de datos utilizados por las aplicaciones. Hay que destacar que la mejora del rendimiento al evitar la contención (valores de T3 frente a T2), la penalización de sobrecarga puede compensarse en pocas iteraciones.

La figura 5 muestra la evaluación del rendimiento para dos escenarios de 20 trabajos cada uno. El eje x representa el nombre de la aplicación y el eje y es el tiempo de ejecución para 100 iteraciones. El umbral utilizado por el monitor para general una alerta de contención fue un uso de memoria superior al 90 %, un porcentaje de fallos de caché de último nivel superior al 40 % y un uso de red superior al 40 % de la capacidad máxima de la red.

El escenario A es un escenario de alto conflicto que consiste en trabajos de las clases A, B_{LL} y F. Para aislar el efecto de interferencia, todas las apli-

caciones son homogéneas. Se debe tener en cuenta que el rendimiento de los dos primeros se degrada en F y que F no se ve afectado por la contención. Para este escenario, el sistema analizó 14 puntos críticos, que produjeron cinco casos de degradación del rendimiento. Para el resto de las aplicaciones, se detectaron todos los casos de rendimiento degradado, excepto el caso *12 (B_{LL})*. En algunos casos, como *15 A*, la aplicación interfiere con varias aplicaciones de F que producen diferentes situaciones de contención. El tiempo total ahorrado al evitar la contención fue de 5.558 segundos, mientras que el coste total (para todas las aplicaciones) de la migración fue de 44,8 segundos para la creación/destrucción de procesos y 223,2 para la redistribución de datos, lo que supone un 4.8 % del tiempo ahorrado.

El escenario B consiste en una carga de trabajo mixta con matrices de entrada con diferentes tamaños. Aquí el número de conflictos fue de 11 y 6 aplicaciones tuvieron una degradación del rendimiento. En total, el tiempo ahorrado fue de 11.016 segundos y la sobrecarga fue de 60,2 y 4.898 segundos para el proceso y la gestión de datos, respectivamente. Esto supone un 45 % del tiempo ahorrado debido a la redistribución de grandes matrices.

IV. Trabajo relacionado

La monitorización de sistemas para plataformas HPC a gran escala es una tarea difícil, que aumenta en complejidad a medida que aumenta la escala de la infraestructura. Nagios [7] es una solución de código abierto para monitorizar redes de hosts y servicios. Es un entorno bien conocido para recopilar información de monitorización en sistemas. Sin embargo, Nagios no está diseñado para sistemas HPC, por lo que su escalabilidad en plataformas de gran escala es incierta.

Desde un punto de vista tecnológico, la monitorización de la información en los sistemas HPC se ha abordado a través de muchos enfoques y herramientas. Un ejemplo de esto es Ganglia [8], que es una de las herramientas de monitorización más utilizadas para los sistemas HPC. Collectd [9], usa un demonio que recopila métricas de rendimiento del sistema (y de la aplicación) y proporciona mecanismos para almacenar los valores de diferentes maneras. Los sistemas distribuidos de gran escala requieren nuevas técnicas, como la programación de períodos subóptimos [10] y el uso de estadísticas del sistema HPC, como TACC Stats in [11], para mejorar la utilización del mismo.

La información del sistema HPC obtenida a través de un servicio de monitorización juega un papel importante en la planificación para compartir recursos y proporcionar computación de alto rendimiento. En [12] se ha descrito un modelo dinámico de arquitectura de meta-planificación para sistemas distribuidos a gran escala basados en monitorización. En este trabajo, el servicio MonALISA se utiliza en combinación con ApMON para recopilar información personalizada y tomar decisiones automatizadas para mejorar la planificación de tareas. La arquitectura de predicción y monitorización de recursos distribuidos descrita en [13] permite encontrar el mejor conjunto de máquinas para ejecutar una aplicación basada en la información recopilada y el resultado de un algoritmo de predicción, que evalúa el rendimiento potencial de un nodo. En este trabajo, presentamos una coordinación más estrecha entre el monitor y el planificador basada en un refinamiento interactivo de la monitorización, que se realiza primero a nivel de todo el sistema y, si es necesario, a nivel de aplicación.

La planificación conjunta de aplicaciones de uso intensivo de memoria y CPU en el mismo nodo que utiliza información de monitorización se ha propuesto en [14] para mejorar la eficiencia energética y el rendimiento general de un supercomputador. Una alternativa diferente son Tetris [15] y LoTES [16], que consideran restricciones en el ancho de banda de la CPU, la memoria, el disco y la red para empaquetar tareas y mejorar la eficiencia del clúster.

V. Conclusiones

Este trabajo tiene como objetivo estudiar algunas de las prioridades de investigación europeas en el área de la tecnología HPC y, más específicamente, en el proyecto ASPIDE. En la solución presentada, los componentes funcionan de manera coordinada y se demuestra que es posible mejorar el rendimiento general de la plataforma y reducir la cantidad de recursos utilizados por el sistema, con el correspondiente ahorro de energía. Aquí, la maleabilidad se emplea no para cambiar el tamaño de la aplicación, sino para migrar las aplicaciones entre diferentes nodos de cómputo. Teniendo en cuenta que en este trabajo asumimos que solo hay riesgo de contención dentro de los nodos. No consideramos la contención de E/S, aunque las soluciones para evitarlo, como la planificación de E/S [17] podrían integrarse en este marco. Como trabajo futuro, planeamos incluir nuevas métricas de rendimiento (como energía o ancho de banda de E/S) en las decisiones del programador y mejorar la ubicación de la aplicación utilizando un análisis más refinado. Todo ello orientado a realizar planificación eficiente que, implícitamente (o explícitamente si se seleccionan políticas para ello) reduzca el consumo energético del clúster sin afectar al rendimiento de los trabajos.

Agradecimientos

Este trabajo está parcialmente subvencionado por el Ministerio de Economía, Industria y Competitividad de España, bajo la ayuda TIN2016-79637-P "Towards Unification of HPC and Big Data Paradigmsz el programa de investigación e innovación Horizon 2020 de la Unión Europea bajo la ayuda No 801091, proyecto "Exascale programming models for extreme data processing" (ASPIDE).

Referencias

[1] Ismail Ari and Ugur Kocak, "Hybrid job scheduling for improved cluster utilization," in *Euro-Par 2013: Parallel Processing Workshops*, 2014, pp. 395–405.

[2] "DaeMon - User Manual," .
[3] Gonzalo Martín, David E. Singh, Maria-Cristina Marinescu, and Jesús Carretero, "Enhancing the performance of malleable MPI applications by using performance-aware dynamic reconfiguration," *Parallel Computing*, vol. 46, no. 0, pp. 60 – 77, 2015.
[4] Gonzalo Martin, David E. Singh, Maria-Cristina Marinescu, and Jesus Carretero, "Towards efficient large scale epidemiological simulations in Epigraph," *Parallel Computing*, vol. 42, pp. 88–102, 2015.
[5] Alberto Núñez, Javier Fernández, Rosa Filgueira, Félix García, and Jesús Carretero, "Simcan: A flexible, scalable and expandable simulation platform for modelling and simulating distributed architectures and applications," *Simulation Modelling Practice and Theory*, vol. 20, no. 1, pp. 12–32, 2012.
[6] "OMNeT++ Discrete Event Simulator," 2019.
[7] "Nagios - The Industry Standard In IT Infrastructure Monitoring," 2018.
[8] Matthew L Massie, Brent N Chun, and David E Culler, "The ganglia distributed monitoring system: design, implementation, and experience," *Parallel Computing*, , no. 7, pp. 817–840, 2004.
[9] "Collectd - The system statistics collection daemon," .
[10] William M. Jones, John T. Daly, and Nathan DeBardeleben, "Application monitoring and checkpointing in HPC: Looking towards exascale systems," in *Proceedings of the 50th Annual Southeast Regional Conference*. 2012, ACM-SE '12, pp. 262–267, ACM.
[11] Todd Evans, William L. Barth, James C. Browne, Robert L. DeLeon, Thomas R. Furlani, Steven M. Gallo, Matthew D. Jones, and Abani K. Patra, "Comprehensive resource use monitoring for hpc systems with TACC Stats," in *Proceedings of the First International Workshop on HPC User Support Tools*, 2014, HUST '14, pp. 13–21.
[12] Florin Pop, Ciprian Dobre, Corina Stratan, Alexandru Costan, and Valentin Cristea, "Dynamic Meta-Scheduling Architecture Based on Monitoring in Distributed Systems," in *2009 International Conference on Complex, Intelligent and Software Intensive Systems*. 2009, IEEE.
[13] S. Rajkumar, N. Rajkumar, and V. G. Suresh, ," in *International Conference on Information Communication and Embedded Systems (ICICES2014)*. 2014, IEEE.
[14] J. Breitbart, J. Weidendorfer, and C. Trinitis, "Case study on co-scheduling for hpc applications," in *2015 44th ICPP Conference Workshops*, Sep. 2015, pp. 277–285.
[15] Robert Grandl, Ganesh Ananthanarayanan, Srikanth Kandula, Sriram Rao, and Aditya Akella, "Multi-resource packing for cluster schedulers," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 4, pp. 455–466, 2014.
[16] Tony T. Tran, Meghana Padmanabhan, Peter Yun Zhang, Heyse Li, Douglas G. Down, and J. Christopher Beck, "Multi-stage resource-aware scheduling for data centers with heterogeneous servers," *Journal of Scheduling*, vol. 21, no. 2, pp. 251–267, Apr. 2018.
[17] F. Isaila, J. Carretero, and R. Ross, "CLARISSE: A middleware for data-staging coordination and control on large-scale HPC platforms," in *16th International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, 2016, pp. 346–355.

Aplicaciones y retos de la sociedad

Implementación SoC de una aplicación de filtrado colaborativo

Francisco Pajuelo Holguera[1] , Juan A. Gómez Pulido[2] y Fernando Ortega Requena[3]

Resumen— **Este artículo propone un sistema de recomendación completo implementado mediante dispositivos FPGA, con el propósito de comprobar las prestaciones de aplicaciones embebidas de bajo consumo energético para el filtrado colaborativo. Aunque en esta implementación el tiempo de computación no es competitivo respecto al obtenido por los microprocesadores multinúcleo, esta propuesta tiene la ventaja de proporcionar una primera aproximación para resolver cualquier problema de predicción basado en el filtrado colaborativo mediante el uso de un entorno de computación ligero y altamente portable. Esta aproximación ha sido probada con éxito utilizando conjuntos de datos habituales para comprobar el rendimiento de los sistemas de recomendación basados en el filtrado colaborativo.**

Palabras clave— **Filtrado Colaborativo, Sistemas de Recomendación, Computación Empotrada, FPGAs, Síntesis de Alto Nivel.**

I. Introducción

Los Sistemas de Recomendación (*Recommender Systems*, RS) [1] son sistemas inteligentes que realizan recomendaciones personalizadas para usuarios de grandes bases de datos. Las recomendaciones se obtienen según el comportamiento de los usuarios cuando manejan la información, lo cual es analizado mediante Analítica de Datos (*Data Analytics*, DA) y Aprendizaje Máquina (*Machine Learning*, ML). Los RS proporcionan recomendaciones según las preferencias de los usuarios, [2] y bloquean la información irrelevante (por lo que también son conocidos como filtros). Los algoritmos desarrollados para RS se centran en propósitos predictivos y se aplican a otros sistemas donde el conocimiento del comportamiento de los usuarios es importante, como por ejemplo, el problema de la predicción del rendimiento académico de los estudiantes [3].

La implementación más popular de los RS es el Filtrado Colaborativo (*Collaborative Filtering*, CF) [4]. CF se basa en la idea de que los usuarios con gustos similares en el pasado, tendrán gustos similares en el futuro [5]. Por ejemplo, si Alicia y Juan han puntuado las mismas películas como positivas, las nuevas películas que Alicia puntúes positivamente, probablemente le gustarán también a Juan. Así, CF se puede aplicar a distintos campos [6]: películas, libros, comercio el electrónico, etc.

CF se construye mediante una matriz que relaciona los usuarios con los ítems de la base de datos. Esta matriz almacena las puntuaciones (explícitas o implícitas) de los usuarios a los ítems, sin necesitar información adicional como las características de los ítems o las propiedades de los usuarios. La matriz de puntuación tiene un alto nivel de dispersión, porque los usuarios solamente puntúan un pequeño número de ítems disponibles. Esta matriz almacena, en bases de datos populares de tamaño medio, cientos de millones de puntuaciones que miles de usuarios han realizado sobre cientos de ítems.

El principal propósito de CF es llenar los huecos de la matriz de puntuaciones [7]. Esta tarea es realizada mediante el algoritmo de Factorización Matricial (*Matrix Factorization*, MF) [8]. Este algoritmo realiza predicciones como una combinación lineal de factores, permitiendo una buena escalabilidad. MF genera un modelo a partir del cual se pueden hacer predicciones [9]. Este modelo predictivo se compone de dos matrices tales que cuando realizamos las predicciones para un determinado usuario y tarea, la fila correspondiente de la primera matriz se multiplica con la columna correspondiente de la segunda. Una característica especial del modelo MF es que las puntuaciones de los usuarios están condicionadas por los K factores latentes que describen los ítems del RS. Por ejemplo, en una base de datos de películas, las puntuaciones de los usuarios están condicionadas por los géneros de las películas que cada usuario puntúa. Si a un usuario le gustan las películas de acción y no las románticas, es muy probable que puntúe positivamente cualquier película de acción y negativamente cualquier romántica. La tarea de MF es encontrar estos factores ocultos a través de la matriz de puntuaciones.

Una de las características más importantes a tener en cuanta en los RS es la gran cantidad de datos involucrados. Las necesidades computacionales para manejar los cálculos asociados a las predicciones pueden ser altas, especialmente si hay requisitos de tiempo real que satisfacer. Por otro lado, el auge de los dispositivos móviles y de bajo consumo limitan la potencia de cálculo de sus procesadores para ciertas aplicaciones, que pueden ser atractivas de instalar en ellos. Por tanto, en este trabajo enfocamos una implementación empotrada (*System on Chip*, SoC) del RS mediante dispositivos FPGA (*Field Programmable Gate Array*). El RS empotrado y de bajo consumo realizado en este trabajo tiene la ventaja de proporcionar una primera aproximación al desarrollo de RS para aplicaciones *off-line* ligeras y de bajo consumo. Esta implementación fue probada con éxito mediante bases de datos utilizadas habitualmente en la literatura para este propósito.

[1]Dpto. de Tecnología de Computadores y Comunicaciones, Univ. de Extremadura, e-mail: franciscoph@unex.es.
[2]Dpto. de Tecnología de Computadores y Comunicaciones, Univ. de Extremadura, e-mail: jangomez@unex.es.
[3]Dep. Sistemas Informáticos, ETSI Sistemas Informáticos, Universidad Politécnica de Madrid, e-mail: fernando.ortega@upm.es.

Un SoC basado en FPGA es un entorno atractivo para esta primera aproximación debido no solo a la aplicabilidad sobre dispositivos ligeros, sino por la combinación de la flexibilidad de la programación software con el rendimiento del hardware para implementar operaciones paralelas, tal como pone de manifiesto la Computación Reconfigurable (*Reconfigurable Computing*, RC) [10].

II. Investigación relacionada

Hay una tendencia actual a utilizar la tecnología de la RC en al ámbito de ML. Alguno de los algoritmos en ML, o partes de ellos, han sido implementados en FPGAs para distintos propósitos, principalmente para tareas de aceleración. Por ejemplo, las Redes Neuronales Convolucionales (*Convolutional Neural Networks*, CNN) [11], el Aprendizaje Profundo (*Deep Learning*, DL) [12], K-Means para Clustering [13][14][15], y la estimación de densidad de núcleos [16] son unos pocos ejemplos en esta dirección. La posibilidad de considerar FPGAs en este contexto ha llevado a empresas como Amazon a ofrecer herramientas y plataformas para acelerar determinadas tareas involucradas en el manejo de grandes bases de datos y servicios en la nube [17].

Algunos trabajos intentan implementar CF y RS sobre FPGAs, considerando distintas partes, enfoques y restricciones. Es más habitual encontrar implementaciones de algunas partes del RS que del RS completo. Por ejemplo, se ha realizado una implementación en FPGA del algoritmo de Gradiente en Descenso Estocástico (*Stochastic Gradient Descent*, SGD) [18] utilizado para entrenar los modelos RS. En este sentido, nuestro trabajo descrito en la Sección IV trata de albergar en una FPGA tanto la modelación del RS como su entrenamiento, incluso manejando bases de datos grandes.

III. Factorización matricial basada en el filtrado colaborativo

La factorización matricial construye un modelo aproximado de matriz de puntuaciones R factorizada en dos matrices [19]: P, que contiene la adecuación de cada factor latente con cada usuario; y Q, que contiene la adecuación de cada factor latente con cada ítem. Los valores de estas dos nuevas matrices son generados a partir de esta matriz de puntuaciones de tal forma que se satisfaga la ecuación (1).

$$R \approx P \cdot Q^T \tag{1}$$

MF ha demostrado su superioridad respecto a otras implementaciones de CF [20] [21] [22]. MF proporciona predicciones precisas (las puntuaciones estimadas) y recomendaciones (el conjunto de los N ítems más relevantes para un usuario). MF también es muy escalable: una vez el modelo es entrenado, las predicciones pueden ser calculadas mediante un sencillo producto de dos vectores de dimensión K. No obstante, este modelo se desactualiza cuando se incorporan al RS nuevas puntuaciones, usuarios o ítmes. Por esta razón, el entrenamiento debe repetirse periódicamente para incorporarlos al modelo, lo cual conlleva un considerable coste computacional.

El proceso de entrenamiento requiere encontrar los valores óptimos de las matrices P y Q que minimizan el error en las predicciones proporcionadas por el modelo. Entrenar los vectores de factores requieren minimizar el error cuadrático regularizado del conjunto de datos de entrenamiento:

$$\min_{p_u, q_i} \sum_{(u,i)\in\kappa} (r_{u,i} - \vec{q}_i^T \cdot \vec{p}_u)^2 + \lambda(||q_i||^2 + ||p_u||^2) \tag{2}$$

En esta ecuación κ es el conjunto de pares (u, i) para los que conocemos la puntuación $r_{u,i}$, p_u y q_i son los vectores de factores latentes del usuario u e ítem i respectivamente, y λ es un hiper-parámetro de regularización que evita el sobre-ajuste.

El modelo se puede entrenar utilizando SGD [23], que de forma cíclica procesa las puntuaciones existentes $r_{u,i}$ hasta alcanzar la convergencia. Para cada caso de entrenamiento (por ejemplo, cada puntuación $r_{u,i}$ conocida) SGD actualiza los valores de las matrices P y Q según las ecuaciones (3) y (4) respectivamente.

$$\vec{p}_u \leftarrow \vec{p}_u + \gamma \cdot (e_{u,i} \cdot \vec{p}_u - \lambda \cdot \vec{q}_i) \tag{3}$$

$$\vec{q}_i \leftarrow \vec{q}_i + \gamma \cdot (e_{u,i} \cdot \vec{q}_i - \lambda \cdot \vec{p}_u) \tag{4}$$

En estas ecuaciones, γ es un hiper-parámetro que controla la tasa de aprendizaje y $e_{u,i}$ es el error en la predicción de la puntuación el usuario u al ítem i:

$$e_{u,i} = r_{u,i} - \vec{q}_i^T \cdot \vec{p}_u \tag{5}$$

El entrenamiento del modelo puede tardar bastante tiempo hasta alcanzar la convergencia. Para reducir este tiempo, se pueden paralelizar las actualizaciones de los vectores de factores de cada usuario e ítem. No obstante, utilizando las ecuaciones (3) y (4) del SGD, la paralelización no es factible, ya que para actualizar cada $\vec{p}_u$ se requiere $\vec{q}_i$, y viceversa. Para afrontar esta dificultad, se puede aplicar la técnica de Raíces Mínimas Alternantes (*Alternating Least Squares*, ALS) [24]. Esta técnica alterna entre fijar el valor de $\vec{q}_i$ y el de $\vec{p}_u$. De esta forma, el sistema calcula cada $\vec{q}_i$ independientemente de los factores de los otros ítems y calcula cada $\vec{p}_u$ independientemente de los factores de los otros usuarios. Esto proporciona la posibilidad de paralelizar masivamente el algoritmo.

Una vez el modelo ha sido entrenado, la predicción de la puntuación del usuario u al ítem i se realiza mediante la ecuación (6). Las recomendaciones a cada usuario pueden obtenerse a partir del conjunto de T ítems no puntuados con las predicciones más altas ($\hat{r}_{u,i}$).

$$\hat{r}_{u,i} = \vec{q}_i^T \cdot \vec{p}_u \tag{6}$$

El Algoritmo 1 muestra el pseudo-código utilizado para MF. Este algoritmo recibe como entrada la matriz de puntuaciones R, el número de factores latentes K y los hiper-parámetros para controlar el proceso de aprendizaje (λ y γ). Por otro lado, el algoritmo devuelve las matrices de factores latentes P y Q, entrenadas a partir de la matriz de puntuaciones. El criterio de convergencia utilizado habitualmente es alcanzar un determinado número de iteraciones.

entradas: R, K, λ, γ
salidas : P, Q
Genera una matriz aleatoria P con U filas y K columnas
Genera una matriz aleatoria Q con I filas y K columnas
repeat
  for *cada usuario u* **do** // Este bucle puede paralelizarse para cada usuario
    for *cada ítem i puntuado por el usuario u* **do**
      $error = R[u][i]$ - dotProduct($P[u]$, $Q[i]$)
      for *cada factor k* **do**
        $P[u][k] += \gamma \cdot (error \cdot P[u][k] - \lambda \cdot Q[i][k])$
      end
    end
  end
  for *cada ítem i* **do** // Este bucle puede paralelizarse para cada ítem
    for *cada usuario u que ha puntuado el ítem i* **do**
      $error = R[u][i]$ - dotProduct($P[u]$, $Q[i]$)
      for *cada factor k* **do**
        $Q[i][k] += \gamma \cdot (error \cdot Q[i][k] - \lambda \cdot P[u][k])$
      end
    end
  end
until *convergencia*
return P, Q

Algorithm 1: Factorización Matricial

IV. Implementación SoC del sistema de recomendación completo

En esta sección se detallan dos implementaciones del sistema de recomendación mediante dispositivos FPGA. La primera implementación (FPGA-RS1) consiste simplemente en instalar los códigos fuentes del RS sobre un sistema operativo empotrado y, a continuación, ejecutar los archivos sobre un microprocesador empotrado en la FPGA. Esta implementación rápida nos permite obtener una primera aproximación para evaluar el rendimiento en relación al tiempo de computación y a la energía consumida, así como comprobar si los resultados obtenidos del algoritmo son los correctos y qué precisión tienen. La segunda implementación (FPGA-RS2) consistió en diseñar un sistema de recomendación paralelizado mediante la programación con las herramientas de síntesis de alto nivel, persiguiendo obtener una mejora del rendimiento respecto a la primera implementación.

A. Conjuntos de datos

Las dos implementaciones S0C del RS fueron testadas utilizando tres conjuntos de datos habituales en la literatura científica para este propósito: bases de datos Movielens-100K, Movielens-1M [25] y *The Movies Dataset* (Kaggle) [26]. Estos conjuntos de datos recogen la actividad de muchos usuarios que puntúan películas con puntuaciones desde 1 hasta 5, donde cada usuario a puntuado al menos 20 películas. La Tabla I muestra las principales características de estos conjuntos de datos.

TABLA I
Conjuntos de datos utilizados para comprobar el funcionamiento de la implementación SoC del RS.

Dataset	Movielens-100K	Movielens-1M	Kaggle
Puntuaciones	100,000	1,000,000	100,000
Usuarios	943	6,000	700
Ítems	1,682	4,000	9,000

B. FPGA-RS1: Implementación de un sistema de recomendación mediante SoC

Para esta primera implementación hemos escogido la tarjeta de prototipado de bajo consumo Zedboard Zynq-7000 (figura 1). Esta plataforma popular de bajo coste incluye un SoC Xilinx Zynq XC7Z020 y todos los elementos necesarios para diseñar cualquier sistema de computación basado en los sistemas operativos Linux, Windows o Android, entre otros. El corazón de la tarjeta es una arquitectura de procesamiento ARM anexa a la lógica programable del SoC. Además, otros elementos proporcionan suficientes características como para interactuar con las necesidades del usuario: interfaces de vídeo HDMI y VGA, entradas y salidas de audio, conector Ethernet, conector para memorias SD e interfaces USB (OTG para manejar periféricos, JTAG para programar el procesador Zynq desde el PC y UART para comunicaciones del puerto serie). La potencialidad del procesamiento es condicionada por los 512 MB de memoria DDR3 y los dos osciladores que generan señales de reloj de 100 y 33,3 MHz.

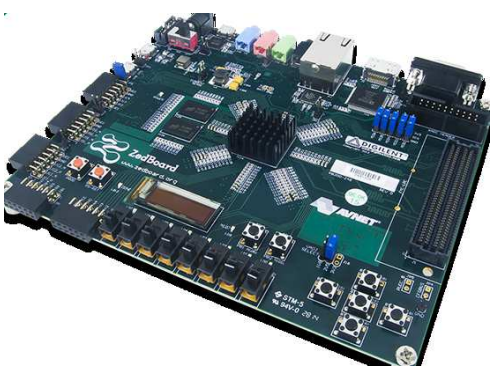


Fig. 1. Tarjeta SoC Zedboard.

Para esta implementación escogimos instalar un sistema operativo Linux (distribución Linaro) sobre la memoria SD, que facilita la ejecución del sistema de recomendación. Este sistema operativo se carga desde una partición separada en la memoria, de forma que los cambios efectuados en el programa se escriben en dicha partición. La ventaja de utilizar la distribución Linaro es que podemos trabajar con la tarjeta de la misma forma que utilizamos habitualmente el microprocesador de un computador personal. Por tanto, los códigos C del sistema de recomendación ejecutado en Zedboard y una CPU son

exactamente los mismos. Una vez instalado Linaro sobre la tarjeta, podemos acceder a él desde el PC a través de un cliente de consola segura; a continuación, transferimos los archivos con los códigos fuente, los compilamos y ejecutamos el archivo con el código máquina de la misma forma que ló haríamos en un PC.

El circuito en la FPGA fue diseñado mediante Xilinx Vivado 2017, el cual incluye el soporte necesario para un sistema de procesamiento basado en Zynq.

C. FPGA-RS2: Implementación de un sistema de recomendación paralelizado

La segunda implementación mejora mucho el rendimiento obtenido por la primera. En este caso, el RS fue paralelizado, por un lado, considerando ALS y estableciendo los bloques de ejecución paralela; y por otro lado, utilizando la herramienta de síntesis de alto nivel (*High-Level Synthesis*, HLS) [27]. Específicamente, para este desarrollo hemos utilizado Vivado HLS [28]. HLS nos permitió diseñar circuitos que fueron paralelizados automáticamente a partir del código C. No obstante, también modificamos el diseño manualmente para incluir distintas directivas de optimización que nos permitieron generar un circuito que proporcionó un rendimiento más alto sin tener que modificar el código C. Estas directivas están relacionadas con el desenrollado de bucles y funciones, y la división de arrays para ejecutar operaciones paralelas.

La figura 2 muestra el pleanteamiento para paralelizar ciertas tareas. De acuerdo con el Algoritmo 1, tras inicializarlo se paralelizan dos bucles consecutivamente, que actualizan las matrices factorizadas correspondientes con cada usuario e ítem. Estos bucles son ejecutados uno tras otro varias veces.

Finalmente, medimos el tiempo transcurrido en el sistema diseñado mediante HLS especificando el mismo dispositivo FPGA y la frecuencia de operación requerida que en el caso de la implementación FPGA-RS1. Una vez el código ha sido sintetizado, HLS nos informa de si la frecuencia de operación dada puede ser soportada por el dispositivo FPGA, así como el número de ciclos de reloj consumidos en el hardware. A partir de este número pudimos calcular el tiempo de ejecución.

D. Resultados y comparación del rendimiento

Las dos implementaciones FPGA y una implementación para CPUs en PC consideran la misma configuración del sistema de recomendación. Por ejemplo, hemos considerado 150 iteraciones según el Algoritmo 1. Esta selección concreta ralentiza la generación del modelo, la cual es la principal causa del tiempo transcurrido en todas las implementaciones.

La tabla II muestra los resultados de rendimiento de ambas implementaciones FPGA y la comparación respecto a una solución basada en CPU, la cual consideró un microprocesador Intel i7-950. Es necesario destacar que esta CPU proporciona una frecuencia de reloj de 3 GHz, mientras que la frecuencia máxima proporcionada por las implementaciones FPGA para ejecutar el RS con seguridad fue de 667 MHz. Los resultados de tiempos mostrados en la tabla son más claros cuando nos fijamos en la "aceleración" en lugar del "tiempo transcurrido". Como la CPU emplea menos tiempo, calculamos su factor de aceleración respecto a las dos aproximaciones en FPGA, simplemente calculando T_{FPGA}/T_{CPU}. Tal como era esperable, las implementaciones SoC son más lentas, aunque este resultado no era el objetivo ni tampoco devalúa las ventajas de implementar un sistema de recomendación en dispositivos empotrados basados en FPGA, tal como se apuntó en la Sección I.

Por último, es importante conocer el impacto energético de ejecutar las implementaciones SoC del RS. Consideramos que el consumo energético es un indicador importante hoy en día, pues un sistema de bajo consumo minimiza los costes operativos cuando se ejecutan muchas predicciones durante mucho tiempo en entornos de computación intensiva, como puede ser el caso de un RS en grandes bases de datos. El consumo de energía el RS en la implementación en CPU fue medido utilizando la herramienta *Powerstat* bajo el sistema operativo Linux Ubuntu, mientras que la herramienta *Power Analyzer* de Xilinx Vivado proporcionó la energía total on-chip de las dos implementaciones FPGA. Es este caso, la reducción del consumo de energía en las implementaciones SoC respecto a la CPU fue muy significativa (87%).

TABLA II
COMPARATIVA DE RESULTADOS DE RENDIMIENTO.

	Time (s)	Power (w)
CPU	113.41	12.31
FPGA-RS1	11,580.68	1.64
aceleración CPU vs FPGA-RS1	×102	
FPGA-RS2	2,934.57	1.64
aceleración CPU vs FPGA-RS2	×26	
aceleración FPGA-RS2 vs FPGA-RS1	×4	
reducción energética FPGA vs CPU	87%	

Los resultados del tiempo transcurrido obtenidos por las implementaciones FPGA del sistema completo RS nos animan a explorar vías de aceleración en tareas específicas del algoritmo.

V. Conclusiones

En este artículo se muestran algunos trabajos que exploran el uso de dispositivos FPGA para implementar diseños de soluciones SoC para los sistemas de recomendación. Con este propósito, hemos diseñado dos implementaciones SoC, una básica y otra más eficiente con procesamiento paralelo, de un sistema de recomendación completo que maneja conjuntos de datos del tamaño habitual en este tipo de sistemas. Estas soluciones proporcionan una primera aproximación para el diseño y evaluación de aplicaciones embebidas para el filtrado colaborativo, muy útiles cuando se consideran entornos de computación *off-line* en dispositivos móviles y de bajo consumo.

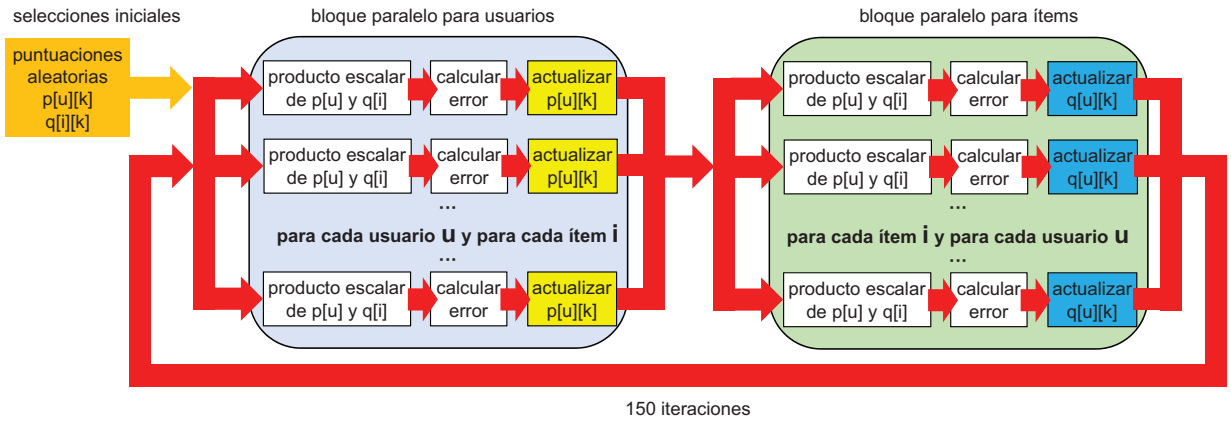


Fig. 2. Estrategia básica para diseñar el sistema de recomendación paralelo.

Agradecimientos

Este trabajo ha sido parcialmente financiado por la Junta de Extremadura, a través del proyecto IB16002, y por el FEDER (Fondo Europeo de Desarrollo Regional, UE) y la AEI (Agencia Estatal de Investigación, España) bajo el proyecto TIN2016-76259-P.

Referencias

[1] D. Jannach, M. Zanker amd A. Felfernig, and G. Friedrich, *Recommender Systems. An Introduction*, Cambridge University Press, 2011.

[2] Jesús Bobadilla, Fernando Ortega, Antonio Hernando, and Abraham Gutiérrez, "Recommender systems survey," *Knowledge-based systems*, vol. 46, pp. 109–132, 2013.

[3] Nguyen Thai-Nghe, Lucas Drumond, Tomas Horvath, Artus Krohn-Grimberghe, Alexandros Nanopoulos, and Lars Schmidt-Thieme, "Factorization techniques for predicting student performance," in *Educational Recommender Systems and Technologies: Practices and Challenges*, pp. 129–153. IGI-Global, 2012.

[4] Gediminas Adomavicius and Alexander Tuzhilin, "Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions," *IEEE transactions on knowledge and data engineering*, vol. 17, no. 6, pp. 734–749, 2005.

[5] Jesús Bobadilla, Francisco Serradilla, and Jesus Bernal, "A new collaborative filtering metric that improves the behavior of recommender systems," *Knowledge-Based Systems*, vol. 23, no. 6, pp. 520–528, 2010.

[6] Hyung Jun Ahn, "A new similarity measure for collaborative filtering to alleviate the new user cold-starting problem," *Information Sciences*, vol. 178, no. 1, pp. 37–51, 2008.

[7] Jonathan L Herlocker, Joseph A Konstan, Loren G Terveen, and John T Riedl, "Evaluating collaborative filtering recommender systems," *ACM Transactions on Information Systems (TOIS)*, vol. 22, no. 1, pp. 5–53, 2004.

[8] Antonio Hernando, Jesús Bobadilla, and Fernando Ortega, "A non negative matrix factorization for collaborative filtering recommender systems based on a bayesian probabilistic model," *Knowledge-Based Systems*, vol. 97, pp. 188–202, 2016.

[9] Steffen Rendle and Lars Schmidt-Thieme, "Online-updating regularized kernel matrix factorization models for large-scale recommender systems," in *Proceedings of the 2008 ACM conference on Recommender systems*, 2008, pp. 251–258.

[10] Russell Tessier, Kenneth Pocek, and Andre DeHon, "Reconfigurable computing architectures," *Proceedings of the IEEE*, vol. 103, no. 3, pp. 332–354, 2015.

[11] Yufei Ma, Naveen Suda, Yu Cao, Sarma Vrudhula, and Jae sun Seo, "Alamo: Fpga acceleration of deep learning algorithms with a modularized rtl compiler," *Integration*, vol. 62, pp. 14 – 23, 2018.

[12] P. R. Gankidi and J. Thangavelautham, "Fpga architecture for deep learning and its application to planetary robotics," in *2017 IEEE Aerospace Conference*, March 2017, pp. 1–9.

[13] J. Canilho, M. VÃ©stias, and H. Neto, "Multi-core for k-means clustering on fpga," in *2016 26th International Conference on Field Programmable Logic and Applications (FPL)*, Aug 2016, pp. 1–4.

[14] F. Winterstein, S. Bayliss, and G. A. Constantinides, "Fpga-based k-means clustering using tree-based data structures," in *2013 23rd International Conference on Field programmable Logic and Applications*, Sep. 2013, pp. 1–6.

[15] Z. Lin, C. Lo, and P. Chow, "K-means implementation on fpga for high-dimensional data using triangle inequality," in *22nd International Conference on Field Programmable Logic and Applications (FPL)*, Aug 2012, pp. 437–442.

[16] Karthik Nagarajan, Brian Holland, Alan D. George, K. Clint Slatton, and Herman Lam, "Accelerating machine-learning algorithms on fpgas using pattern-based decomposition," *Journal of Signal Processing Systems*, vol. 62, no. 1, pp. 43–63, Jan 2011.

[17] "Amazon ec2 f1 instances," 2019.

[18] K. Kara, D. Alistarh, G. Alonso, O. Mutlu, and C. Zhang, "Fpga-accelerated dense linear machine learning: A precision-convergence trade-off," in *2017 IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, April 2017, pp. 160–167.

[19] Yehuda Koren, Robert Bell, and Chris Volinsky, "Matrix factorization techniques for recommender systems," *Computer*, vol. 42, no. 8, pp. 30–37, 2009.

[20] Andriy Mnih and Ruslan R Salakhutdinov, "Probabilistic matrix factorization," in *Advances in neural information processing systems*, 2008, pp. 1257–1264.

[21] Yehuda Koren, "Factorization meets the neighborhood: a multifaceted collaborative filtering model," in *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2008, pp. 426–434.

[22] Fernando Ortega, Antonio Hernando, Jesus Bobadilla, and Jeon Hyung Kang, "Recommending items to group of users using matrix factorization based collaborative filtering," *Information Sciences*, vol. 345, pp. 313–324, 2016.

[23] Leon Bottou, "Large-scale machine learning with stochastic gradient descent," in *Proceedings of 19th International Conference on Computational Statistics*. 2010, pp. 177–186, Springer.

[24] R. M. Bell and Y. Koren, "Scalable collaborative filtering with jointly derived neighborhood interpolation weights," in *Seventh IEEE International Conference on Data Mining (ICDM 2007)*, Oct 2007, pp. 43–52.

[25] F. Maxwell Harper and Joseph A. Konstan, "The movielens datasets: History and context," *ACM Trans. Interact. Intell. Syst.*, vol. 5, no. 4, pp. 19:1–19:19, Dec. 2015.

[26] R. Banik, "The movies dataset, version 7," 2017.

[27] J. Cong, B. Liu, S. Neuendorffer, J. Noguera, K. Vissers, and Z. Zhang, "High-level synthesis for fpgas: From prototyping to deployment," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 30, no. 4, pp. 473–491, April 2011.

[28] D. O'Loughlin, A. Coffey, F. Callaly, D. Lyons, and F. Morgan, "Xilinx vivado high level synthesis: Case studies," in *25th IET Irish Signals Systems Conference 2014 and 2014 China-Ireland International Conference on Information and Communications Technologies (ISSC 2014/CIICT 2014)*, June 2014, pp. 352–356.

Integración de DVS en sistema empotrado basado en FPGA para clasificación visual de alta velocidad mediante acelerador de CNNs

A. Linares-Barranco[1], A. Ríos-Navarro, R. Tapiador-Morales[2], C. Amaya, G. Jimenez.

Resumen— **El Deep-learning está hoy día entre los temas más influyentes en torno a la informática y al hardware dedicado. Su aplicación se expande continuamente a nuevos ámbitos. Para aplicaciones de visión, las Redes Neuronales por Convolución (CNN) están alcanzando unas tasas de acierto muy significativas para tareas de clasificación. Últimamente han surgido numerosos aceleradores hardware para CNNs que mejoran a las soluciones basadas en *clusters* de CPUs y GPUs. Esta tecnología suele probarse en FPGA mediante prototipos para contrastar los resultados de forma previa a una producción masiva en ASIC. El uso de cámaras convencionales (25/30 fps) limita considerablemente la capacidad de estos sistemas para aplicaciones de alta velocidad. Sin embargo, el uso de retinas de silicio *(Dynamic-Vision-Sensors - DVS)*, que emulan el comportamiento de parte de las células de la retina biológica, está en creciente interés en este campo. Su naturaleza pulsante permite representar la información de una manera continua, sin pausas ni tiempos de integración, de manera que el flujo de impulsos o eventos es muy adecuado para tareas de alta velocidad. En este artículo se presenta una descripción VHDL/HLS de un diseño en pipeline para FPGA que recolecta eventos para crear histogramas normalizados, los cuales se presentan a un acelerador de CNN, llamado NullHop, que ejecuta una CNN que juega al "Piedra-Papel-Tijera". El sistema responde en 6ms al símbolo presentado, siendo superior a los 60ms que un ser humano tiene de media como umbral para percibir engaño sobre la realidad del juego. Se ha usado HLS para describir los bloques de computación para la normalización y VHDL para describir el circuito de creación de histogramas y normalización usando técnicas de pipeline, replicación de unidades de computación y memorias de doble buffer. Se ha obtenido un 67% de aceleración (con una tasa máxima de procesamiento de 167fps) respecto del mismo sistema hardware que ejecuta en software la recolección y normalización de histogramas.**

Palabras clave— **Deep-learning, CNN, DVS, AER, Neuromórfico, FPGA, sistema empotrado.**

I. Introducción

LAS *Field-Programmable-Gate-Arrays (FPGA)* han demostrado un alto impacto no solo en el prototipado de arquitecturas para *deep-learning*, si no también en productos comerciales en este campo. Para aplicaciones visuales, las redes neuronales por convolución (CNNs) representan una de las aproximaciones más utilizadas en un creciente número de aplicaciones de visión artificial de gran envergadura [1–3]. Las CNNs tienen una arquitectura típica compuesta por múltiples capas de neuronas, cuyos pesos se disponen como matrices de convolución, que extraen características de las imágenes de entrada. La primera capa está compuesta por un número de filtros de convolución que extraen características básicas de la entrada (detección de bordes, líneas, orientaciones, ...). Las siguientes capas, cada vez más profundas, también se componen de filtros de convolución, los cuales van tomando como entrada las salidas de las capas anteriores para combinarlas y extraer características más complejas. Los pesos utilizados en estas operaciones de convolución se obtienen tras un complejo y tedioso método de entrenamiento del sistema. La operación básica en una capa de una CNN es la expresada en (1), donde $F^{out}_{(a,b)}$ es un píxel de salida de una convolución aplicada a una características de entrada F^{in} con un *kernel* C. Para obtener la característica de salida final, las características de entrada elegidas son procesadas con sus *kernels* correspondientes y acumuladas en las características de salida. Es normal encontrar operaciones no lineales en la conexión entre capas, como *ReLu (Rectification Unit)*, y operaciones de submuestreo para reducir el dominio del problema a un conjunto más cerrado de características complejas que puedan ser clasificadas por una *fully-connected-layer* (clasificador neuronal clásico) al final de la arquitectura.

$$F^{out}_{(a,b)} = \sum_{i=-N_{in}/2}^{N_{in}/2} C_{i,j} * F^{in}_{(a-i/2,b-i/2)} \quad (1)$$

Su núcleo computacional tan simple unido a la técnica de aprendizaje supervisado, las han convertido en el método más elegido para extracción de características de imágenes a un nivel semántico elevado y válido para clasificación, localización y tareas de detección visual [4].

Las CNNs se entrenan típicamente mediante la técnica de *backpropagation*, usando un conjunto de ejemplos etiquetados de un tamaño relativamente grande. Este conjunto permite que la CNN configure sus pesos de forma adecuada para inferir la clase correcta ante una imagen de entrada que no se ha usado durante el entrenamiento. Las CNNs normalmente son entrenadas en plataformas hardware, como las *graphical processing units (GPUs)*, o arquitecturas especializadas, como en [5], de forma previa a su uso en una aplicación en particular. El proceso de entrenamiento es muy pesado, computacionalmente, y requiere de varias iteraciones y de un *dataset* de entrenamiento muy bien creado y configurado para ex-

[1]Los autores pertenecen al Dpto. de ATC, Univ. Sevilla, e-mail contacto: alinares@atc.us.es

[2]Soportado por el programa de Formación de Personal Investigador de la Universidad de Sevilla

traer la mejor tasa de acierto de una arquitectura CNN prefijada.

Una vez entrenada, la CNN se puede usar para la tarea destino. Según su arquitectura, la ejecución puede tener una carga computacional alta. Las arquitecturas de mayor impacto en el estado del arte en los últimos años requieren algunos miles de millones de operaciones *MAC (multiply-accumulate)* por imagen. Por tanto, ejecutar estos algoritmos en dispositivos móviles, con aceleradores gráficos potentes, suponen un alto coste en consumo y en muchas ocasiones no adecuadas para aplicaciones de alta velocidad.

Por ejemplo, en [6] el modelo *Inception V3* para reconocimiento de objetos tiene una tasa de acierto del 78% con 27M de parámetros y 12GOP, consumiendo 51ms para procesar un fotograma en el procesador para móviles *Kirin 980* con acelerador neuronal en hardware [1]. La red *Inception ResNet V1* para reconocimiento de caras tiene una tasa de éxito del 98% y necesita alrededor de 113ms en el mismo chip. Y la arquitectura *ICNET* para segmentación semántica, usada en conducción autónoma, tiene una tasa de éxito del 70% siendo posible su ejecución en tiempo real (30fps) cuando se instalan potentes GPUs en el vehículo comercial, aunque requiere 275ms por fotograma con el chip *Kirin 980*. ICNET requiere de 64ms cuando se ejecuta en una *Free-TPU* para FPGA. Estos ejemplos resaltan que las soluciones basadas en FPGA para resolver problemas reales están cada vez más cerca de ser una realidad comercial.

Sin embargo, el límite de los 30fps de las cámaras comerciales convencionales puede ser un costoso problema que resolver si se requiere instalar cámaras de alta velocidad en los vehículos, debido a las limitaciones de acceso a memoria e interfaces a computador de las versiones comerciales actuales. La ingeniería neuromórfica ha venido trabajando durante más de 20 años en sensores bio-inspirados (conocidos como *Dynamic-Vision-Sensors, DVS*) [7, 8] que ofrecen tasas equivalentes a $\approx 20kfps$ comparado con cámaras basadas en CCD gracias a no trabajar con imágenes, sino con eventos (como impulsos nerviosos). Las retinas DVS permiten a cada píxel trabajar de forma independiente a los demás, pudiendo enviar la actividad registrada localmente de forma inmediata. La mayoría de estos sensores monitorizan los cambios locales de luminosidad a nivel de píxel, de tal forma que se comportan como una neurona y envían un impulso o evento cada vez que detectan un cambio por encima de un umbral configurable. Normalmente, estos eventos son enviados con una polaridad que indican si el cambio de luminosidad ha sido creciente o decreciente. En la Fig 6 (rectángulo monitor en la pantalla del PC) se muestra un histograma 2D de 2k eventos capturados durante 6ms en este caso particular. Se puede ver como los eventos positivos (blancos) representan la parte frontal de la mano en movimiento, mientras que los eventos negativos (negros) representan la parte trasera de la mano. Cuanto más rápido sea el movimiento de la mano, menor será el tiempo requerido para recoger los 2k eventos. Si estos histogramas pudieran ser usados por las CNNs, tendríamos *fps* dinámicos que se adaptan a la velocidad del escenario particular donde se requiera detectar o clasificar un objeto.

En este artículo comparamos el rendimiento de la recolección de eventos en un histograma y su normalización en FPGA, usando un pipeline con replicación de unidades de computación creadas en HLS, respecto a una versión previa del sistema donde ambas tareas se realizaban en software en los ARM empotrados en la FPGA (Zynq 7100).

El artículo está estructurado de la siguiente forma: El apartado II introduce el acelerador de CNNs *NullHop* y las consideraciones para su prototipado en FPGA. El apartado III describe detalladamente el circuito diseñado en VHDL y HLS para recolectar y normalizar histogramas de eventos que serán evaluados en el NullHop. El apartado IV muestra resultados de rendimiento y la comparativa respecto a la alternativa software para el ejemplo de CNN llamado *RoShamBo* que permite a un usuario jugar al "piedra-papel-tijera" contra la CNN.

II. Acelerador de CNNs NullHop

Este acelerador aprovecha el escaso mapeo que tienen las CNNs a partir de ciertas capas de convolución debido, en parte, a las funciones no-lineales como ReLU para adaptar los resultados de una capa a la siguiente. Estas operaciones implican que el número de valores a cero de los mapas de características entre capas sea creciente conforme se profundiza en la arquitectura de la CNN. Este acelerador, llamado *NullHop* [9], detecta y evita los cálculos con los ceros tanto de los *kernels* de convolución como de los mapas de características de entrada de una capa de la CNN. Además utiliza un esquema de compresión que optimiza la ejecución de estas capas ya que reduce los accesos a la memoria externa y es más eficiente que otros esquemas de codificación [10]. De forma similar a otros aceleradores de CNNs [10–18], *NullHop* utiliza un procesamiento por etapas configurable con 128 unidades de computación (MACs) que mantienen una alta eficiencia para un rango de tamaños de *kernels* de convoluciones y número de mapas de características.

A. Arquitectura

La Fig. 1 muestra un diagrama de bloques del acelerador *NullHop*. Su interfaz permite alimentarlo con un bus de datos de 32-bits, configurarlo con un bus separado, y controlarlo mediante algunas líneas de reloj, *reset* y *handshake*. Este acelerador implementa una capa de convolución con 128 unidades MAC, seguidas de una transformación *ReLU* y una operación de *MAX-pooling* que pueden ser usadas opcionalmente. La CNN se implementa capa a capa. Por tanto, para ejecutar una capa, hay que progra-

[1] Datos tomados de http://ai-benchmark.com/ranking_all_2_1_2.html on 5th May 2019

mar los mapas de características de entrada y los valores de los *kernels*, que se almacenan en dos bloques SRAM independientes accesibles por las interfaces de datos y de configuración respectivamente.

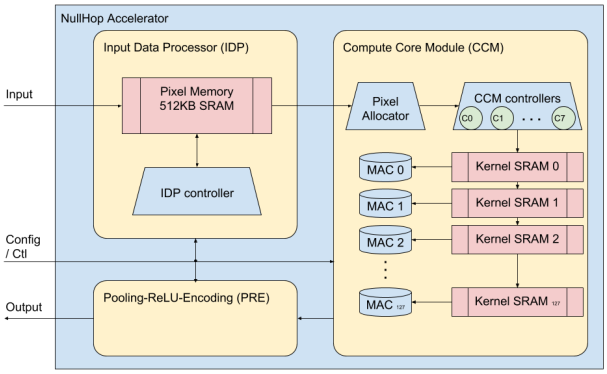


Fig. 1: Diagrama de bloques del acelerador de CNNs NullHop. figure

Los mapas de características producidos por la capa actual son enviados secuencialmente, píxel a píxel, por la interfaz de salida para ser almacenados en una memoria externa (DDR conectada al procesador y accedida por DMA). Estos datos serán volcados nuevamente en el acelerador para la ejecución de la siguiente capa de la CNN una vez enviados los nuevos parámetros al acelerador. Los mapas de características siempre se almacenan en formato comprimido, el cual no se descomprime, pero se decodifica durante la computación.

NullHop usa una técnica de compresión de matriz dispersa que produce un nivel medio de compresión más alto que otros métodos, como [10]. Y es más fácil de decodificar que la codificación de Huffman usado en [14]. En esta codificación se usan *Sparsity Maps* (**SM**), que es una máscara 3D con el mismo número de entradas que píxeles en el mapa de entrada; y una lista de valores no nulos *(Non-Zero Value List - **NZVL**)*. El SM se usa para reconstruir las posiciones de los píxeles no nulos presentes en el NZVL.

El procesador de decodificación de entrada *(Input Decoding Processor - **IDP**)* lee una parte de la entrada comprimida para trasladar los valores no nulos al módulo de computación de cores *(Compute Core Module - **CCM**)*. Por tanto, el IDP se está saltando los valores nulos de los mapas de características de entrada comprimidos, evitando realizar MAC innecesarias y ahorrando tiempo de computación. Además de los valores de los píxeles, el IDP indica las posiciones de los píxeles (fila, columna, índice dentro del mapa de entrada) al CCM. El asignador de píxeles *(Pixel Allocator)* (en el CCM) va asignando la computación de los píxeles a través de los controladores (*Controllers*) disponibles. Cada *Controller* planifica las operaciones para un subconjunto de unidades funcionales (MAC) y envía las peticiones de lectura correspondientes a la SRAM de los *kernels*. Todos los bloques MAC de un *Controller* reciben el mismo píxel de entrada de su *Controller*, pero con pesos de diferentes *kernels*, generando píxeles en mapas de características de salida diferentes. Estos resultados son enviados de forma opcional a un transformador ReLU y a una etapa de submuestreo (max-pooling) de forma previa a ser enviados al bloque de compresión (módulo PRE en la figura). El chip envía a la memoria externa los mapas de características comprimidos. Este acelerador *NullHop* se ha descrito en *SystemVerilog* y se ha sintetizado para FPGA y para ASIC, como se describe en detalle en [9].

B. Implementación en FPGA

La descripción *SystemVerilog* del *NullHop* se ha adaptado para ser sintetizado en el PSoC de Xilinx Zynq 7100 disponible en la plataforma MMP de AvNet. Este PSoC está formado por un computador ARM dual-core, llamado *processing system (PS)*, y por una lógica programable tipo Kintex-7, denominada *programmable logic (PL)*. Ambos sistemas se conectan usando el bus *AXI4-Stream* con *Direct Memory Access (**DMA**)*, un protocolo libre (open source) que conecta el PL con el PS. En el PS se ejecuta una versión de Linux (Petalinux) para controlar al acelerador. Este maneja las lecturas y escrituras entre la memoria DDR de la placa MMP y la memoria de bloque de la FPGA (BRAM). El PS se encarga también de ejecutar la última fase de la CNN, que es un clasificador neuronal, conocido como *Fully-Connected-Layer (FCL)*. En el SO se ha incluido un módulo USB-host para poder recibir eventos de una retina DVS externa para construir los histogramas por software. La retina usada es la DAVIS240C (comercializada por IniVations[2]) [8] para las demostraciones en tiempo real que se usan para medir los resultados de este trabajo. En este caso, el SO ejecuta también el software cAER [3], un *framework* abierto para leer y procesar información de estas retinas en C/C++.

La Fig 2 muestra el diagrama de bloques de la arquitectura global del sistema en la FPGA, el cual incluye los bloques MM2S (*Memory Mapped To Stream*) y S2MM (*Stream to Memory Mapped*) utilizados como interfaces entre el acelerador y el bus AXI Stream. MM2S y S2MM incluyen FIFOs para transmitir los datos y máquinas de estados finitas *(finite-state-machines FSM)* para adaptar el protocolo entre el bloque IP del AXI-DMA de Xilinx y la interfaz del *NullHop*. Permite configurar parámetros como la longitud de las ráfagas y controlar las interrupciones.

Para cada capa, el PS carga o localiza en memoria DDR los *kernels* y los mapas de características de entrada. Entonces inicia una serie de transferencias DMA, controladas por interrupciones, para volcar los datos y parámetros en el acelerador y al mismo tiempo para leer los mapas de características de salida en otra zona de la DDR externa. Mientras tanto, el PS queda libre para realizar otras tareas, como la obtención del siguiente histograma y su normalización usando el software cAER mencionado

[2]https://inivation.com/dvs/
[3]https://github.com/inivation/caer

anteriormente.

La Fig. 3 muestra la jerarquía de memoria desde la aplicación de usuario hasta el acelerador en FPGA. Hay dos mecanismos para trabajar en sistemas empotrados bajo Linux: (1) *user-level*: la función *mmap()* se usa para mapear un segmento del espacio de memoria físico del dispositivo en el espacio virtual de memoria del procesador. Esta función es invocada directamente por la aplicación y las transferencias DMA se pueden configurar por encuesta *(polling)*, quedando la aplicación de usuario bloqueada con frecuencia mientras espera a que la transferencia de datos se complete; o (2) *kernel-level*: un rutina se ejecuta en un nivel superior de privilegio del SO, con soporte de interrupciones, manteniendo la aplicación de usuario libre de estados de bloqueo mientras se completan las transferencias de datos. En principio, el nivel de *kernel* es más seguro, al tener mayor nivel de privilegio que el nivel de usuario, evitando un posible mal uso de la memoria física reservada a otros procesos del SO.

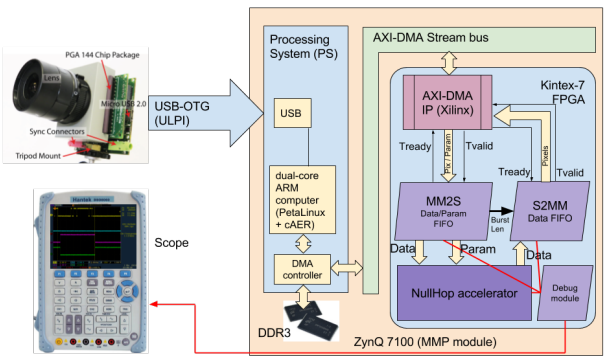


Fig. 2: Diagrama de integración del NullHop en PSoC.
figure

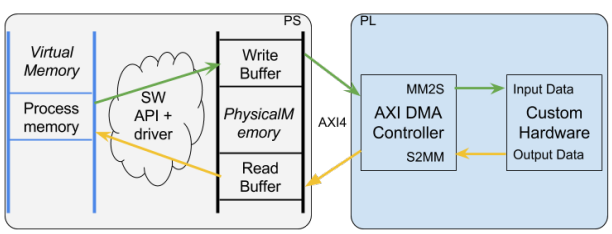


Fig. 3: Jerarquía de memoria para DDR en PSoC con SO. La aplicación de usuario trabaja en un espacio virtual, mientras que el controlador DMA en PL trabaja con espacio físico. La API y/o el driver realiza las transferencias a/desde ambos espacios.
figure

El uso de cAER para la integración de la DVS en el sistema y la ejecución de las rutinas de creación y normalización de histogramas están requiriendo una carga computacional considerable en el PS que afecta tanto a la latencia del sistema en procesar un histograma como al consumo de energía del sistema. En este trabajo presentamos una modificación del sistema que realiza el histograma y su normalización en la FPGA, liberando al PS del cálculo de instrucciones de punto flotante, del uso del cAER y del interfaz USB-host. El diseño se ha hecho para la retina PAER-DVS, que cuenta con una resolución de 128x128 píxeles y utiliza el bus AER para transmitir los eventos de los píxeles estimulados con una menor latencia que cuando se usa USB. Al tener esta retina más nivel de ruido, en el montaje completo se ha incluido un filtro de actividad de *background*.

III. Histogramas de eventos y Normalización

En esta sección se expone el circuito creado para generar un histograma normalizado de una retina DVS en la parte FPGA del sistema empotrado. Para ello se ha usado VHDL para describir el comportamiento del sistema y HLS para describir los bloques de operaciones necesarias para la normalización. Los histogramas son almacenados en la memoria interna de la FPGA (BRAM) desde su recolección hasta su compresión a formato *NullHop* pasando por el proceso de normalización. Para mantener una transferencia de datos continua al *NullHop* se ha planteado un esquema de doble *buffer*. La normalización requerida para los histogramas ha de ser la misma que se usó para crear el *dataset* de entrenamiento de la CNN, el cual responde a las fórmulas presentadas en (2).

$$
\begin{aligned}
S &= \textstyle\sum_{a=0}^{N}\sum_{b=0}^{N} F_{in}(a,b) \\
c &= \textstyle\sum_{a=0}^{N}\sum_{b=0}^{N} f(F_{in}(a,b)) \\
f(X) &= \begin{cases} 0 \text{ ,if X=0} \\ 1 \text{ ,otherwise} \end{cases} \\
mean &= S/c \\
\sigma &= \sqrt{\frac{\sum_{a=0}^{N}\sum_{b=0}^{N}[F_{in}(a,b)-mean]^2}{c}} \\
F_{norm}(i,j) &= \frac{F_{in}(i,j)+3\sigma}{6\sigma}, \forall i,j \in [0,N]
\end{aligned}
\tag{2}
$$

Donde S es la suma de todos los valores no nulos de las características de entrada F_{in} (features IN), c es el número de píxeles con valores no nulos, *mean* es el valor medio para los píxeles no nulos, σ es la raíz cuadrada de la varianza, que es usado para normalizar los píxeles (F_{norm}) del histograma que se presenta a la CNN.

La Fig. 4 muestra el diagrama del circuito propuesto. Los eventos AER de la retina PAER-DVS se van añadiendo al doble *buffer* implementado en BRAM, llamados (DVSmem1, SMarray1) y (DVSmem2, SMarray2). *DVSmem* almacena los histogramas de eventos, mientras que *SMarray* almacena una máscara de los píxeles no nulos. Una FSM está configurada para recolectar un número fijo de eventos (2K eventos en este caso) en la BRAM, correspondientes como un histograma (HT). La FSM cambia de *buffer* en BRAM cuando completa el número de eventos a recolectar. Mientras se van recolectando eventos para el segundo *buffer*, el sistema procesa

el primero realizando la normalización ((2)) del histograma, su conversión a formato comprimido y su transmisión al acelerador. Hasta que estas tres tareas no hayan finalizado, la FSM que recolecta eventos no podrá volver a usar este *buffer* y se quedaría a la espera si fuera necesario.

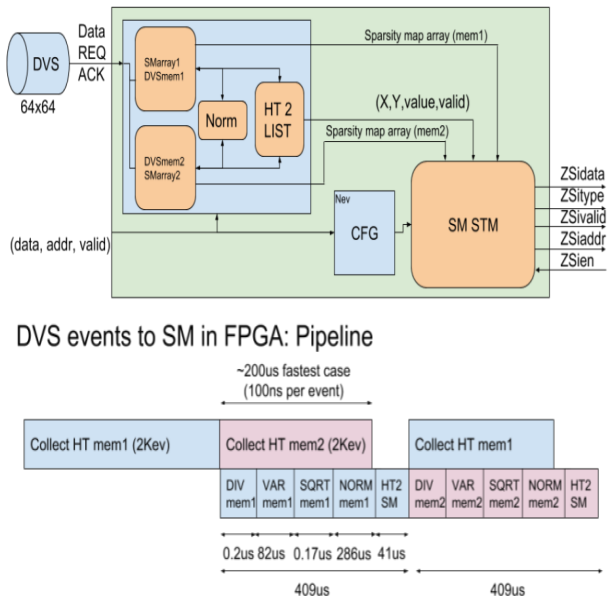


Fig. 4: Diagrama del circuito de doble *buffer* para preparar histogramas para *NullHop* (top) y esquema pipeline de ejecución de recolección, normalización y compresión (bottom).
figure

De acuerdo a la formulación de normalización, se han necesitado varias unidades de computación para realizar las operaciones requeridas. Estas unidades se han descrito usando HLS de Vivado. Las unidades necesarias son: división, raíz cuadrada, varianza y la propia normalización. En lugar de usar operaciones de punto flotante, se han descrito las unidades usando punto fijo con notación Q24.16 [4] para los cálculos internos, y Q16.8 para la normalización final. Una vez que la normalización ha concluido y el resultado se encuentra almacenado en BRAM, una última FSM convierte su contenido en secuencias de mapas dispersos (SM) codificados para *NullHop*. En la Fig. 4 se puede apreciar el tiempo requerido por cada módulo HLS y la última FSM, el cual es mayor que el tiempo requerido para recolectar los 2k eventos cuando la DVS trabaja a su máxima capacidad.

Para poder extraer el máximo rendimiento se han implementado tantas instancias de los bloques HLS como han sido necesarias para reducir la latencia de normalizar un histograma. Por ejemplo, como se muestra en la Fig. 5, el bloque NORM se ha replicado 22 veces, de forma que cada bloque recibe un píxel de BRAM en cada ciclo de reloj y de forma consecutiva. Tras alimentar los 22 bloques, el primer bloque NORM se queda libre y permite tomar un nuevo píxel. Tras esperar la latencia del primer píxel (que se corresponde con la latencia del bloque

[4]Qn.m means n bits for the integer part of the number and m bits for the decimal part

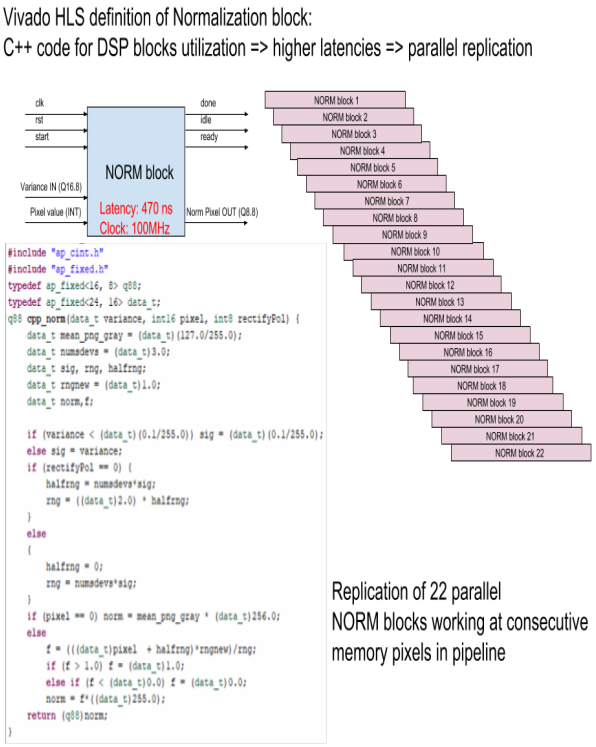


Fig. 5: Bloque HLS para la fase de NORM. Se instancian 22 unidades en paralelo.
figure

NORM[5]), los demás resultados se obtienen a razón de uno por ciclo de reloj. La figura muestra también el código C del bloque.

Para histogramas de 64x64 píxeles de resolución, reducidos a la mitad de la resolución de la DVS para poder ser usados en la demostración *Roshambo* (piedra-papel-tijera), se necesita medio microsegundo para normalizar y convertir los histogramas en mapas dispersos codificados para alimentar el *NullHop*. Esto es alrededor de cuatro veces más rápido que la solución anterior con cAER ejecutándose en los núcleos del ARM, tal como se demuestra en la siguiente sección de resultados.

IV. Experimentos y resultados

Para medir el rendimiento de la solución propuesta se ha testado el diseño en un escenario donde la normalización de los histogramas fue previamente realizada en C++ (en cAER) ejecutándose desde el SO Petalinux en el PS del PSoC de Xilinx de la plataforma MMP. En este escenario se ejecuta una CNN en el sistema que es capaz de clasificar un símbolo presentado a la retina con la mano de un usuario. El símbolo a clasificar puede ser piedra, papel o tijera. El software ejecutándose en el PS mapeará el símbolo clasificado por el acelerador de CNN para presentar al usuario el opuesto según el juego. Dado que el sistema requiere unos 10ms por histograma, se puede incluso procesar varios y tomar la media histórica en menos de 60ms, que es el tiempo en que un humano medio no detecta engaño. La

[5]470ns para CLK de 100MHz y 783ns para CLK 60MHz

CNN se ha diseñado para esta tarea de clasificación de forma que pueda extraer el máximo rendimiento del acelerador *NullHop*, tal como se describe en [9].

La Fig. 6 muestra el escenario de test. Una retina DVS con salida AER paralela se conecta a una placa AER-Node [19], que monta una OKAERtool [20] , programadas con un monitor de eventos USB y un *Background Activity Filter (BAF)* [21], lo cual nos permite visualizar en la pantalla del ordenador el histograma de la retina (parte derecha). La salida del filtro BAF se conecta a una USBAERmini2 [22], la cual monitoriza los eventos al tiempo que los retransmite a la placa MMP. La salida del BAF puede verse en la parte derecha de la pantalla del ordenador. La MMP está ejecutando la CNN para jugar al juego de piedra-papel-tijera. La salida del *NullHop* se usa para encender una barra de LEDs indicando cuál es la clase ganadora respecto del símbolo presentado. En esta figura se aprecia que la mano presenta el gesto de papel a la DVS y la barra de LEDs indica tijera para ganar a su oponente.

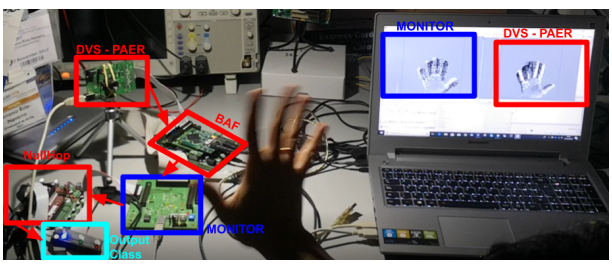


Fig. 6: Escenario del juego piedra-papel-tijera con DVS y NullHop.
figure

La CNN diseñada para este escenario consta de 5 capas de convolución y un clasificador neuronal que es ejecutado en los ARMs. Cada capa de convolución es ejecutada de una vez en el acelerador, una vez cargados los pesos de los *kernels* y los mapas de características de entrada. Por tanto, el software ejecutándose en los ARMs necesita iterar 5 veces el traspaso de datos con el acelerador mediante la interfaz AXI-DMA, teniendo ésta una alta repercusión en el rendimiento del sistema, tal como se describe en [23]. El tiempo mínimo sostenido que se requiere para procesar un histograma es de 6 ms en este escenario. Este tiempo se obtiene cuando el movimiento de la mano es suficientemente rápido como para producir los 2k eventos en menos de 6 ms.

Este controlador software se ha modificado convenientemente para permitir al sistema recibir la entrada de la primera capa de convolución del circuito implementado en la FPGA en lugar de recibirlo por el DMA. Por tanto, con esta solución, no se requiere el uso del cAER, ni de la interfaz USB-host, lo que decrementa considerablemente el consumo de los ARMs.

La Fig. 7 muestra dos capturas de osciloscopio del *NullHop* ejecutando la red de ejemplo comentada, con un reloj de 60MHz para la FPGA y 800MHz para los ARM. La captura superior se ha tomado del sistema inicial, incluyendo cAER y la normalización

TABLA I: Consumo de recursos de FPGA para el NullHop, circuito de normalización (DVS2SM), AXIDMA y logica de depuración.
table

Zynq7100 (#)	LUT (277K)	FF (555K)	BRAM (755)	DSP (2020)
DVS2SM	**50**	**25**	**4**	**513**
NullHop	**210**	**107**	**385**	**144**
AXIDMA	**5**	**6**	**11**	**0**
Debug	**1**	**1**	**0**	**0**
Total	**266**	**139**	**401**	**657**

por software de los histogramas. La señal *NH_idle* indica si el acelerador está computando ('0') o en reposo ('1'). Se ha resaltado la secuencia de las 5 capas de convolución. La latencia para procesar un histograma en este escenario es de $\approx 10ms$. La segunda captura muestra el mismo comportamiento cuando se usa el circuito propuesto para la normalización de los histogramas y usando el mismo número de eventos para cada histograma (2k eventos). Se puede apreciar como la actividad de la DVS se detiene cuando el sensor percibe una alta actividad visual. En este caso, el tiempo que requiere el sistema por cada histograma y su clasificación en el *NullHop* se reduce a $\approx 6ms$. Esto implica una aceleración del 67% según la ley de Amdhal. Además, al usar este circuito en la FPGA, el sistema también mejora gracias al comportamiento en pipeline de las etapas de (1) recolección de eventos, (2) normalización de histogramas y (3) ejecución iterativa de las capas de la CNN en el acelerador *NullHop*; tal como se muestra en la parte baja de la Fig. 7, donde se aprecia cómo los histogramas normalizados esperan en BRAM antes de ser enviados al acelerador mientras se completa la computación del histograma anterior.

En la tabla I se muestran los recursos disponibles en la Zynq 7100 en la primera fila divididos en LUTs, bloques de flip-flops, memoria BRAM y bloques DSP. Las siguientes filas presentan los recursos consumidos por cada parte del sistema. Se puede apreciar cómo el uso de HLS para implementar la normalización ha aumentado considerablemente el número de bloques DSP comparado con la versión anterior (cAER) que requería únicamente de 144 bloques DSP para implementar los 128 MACs y otras operaciones del IDP y el PRE. También se puede apreciar un incremento del número de recursos LUT, FF y BRAM para este circuito de normalización.

Respecto al consumo de energía, se ha medido un consumo de 272mW en el PL y 1W en el PS ejecutando la versión cAER de la demo. Para la versión que ejecuta el circuito de normalización en FPGA, el consumo total medido durante la ejecución del juego es de 500mW incluyendo tanto el PL como el PS.

V. Conclusiones

Lo sensores neuromórficos de visión, como el DVS permiten, en las aplicaciones de deep-learning, sobrepasar los límites de las cámaras convencionales de 30fps en aquellas aplicaciones que lo requieran.

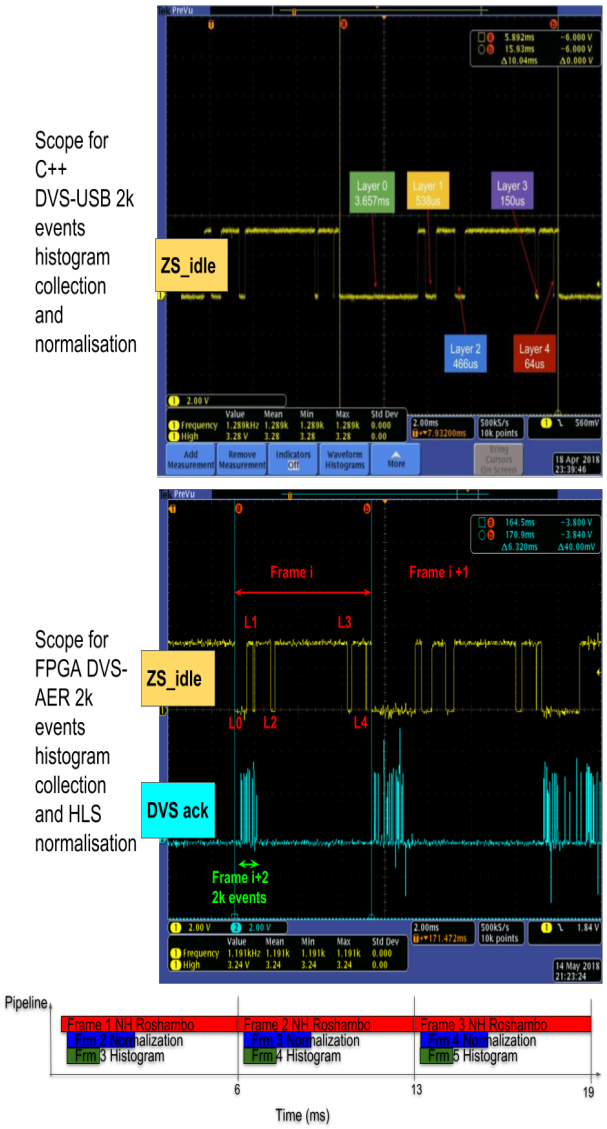


Fig. 7: Capturas de osciloscopio del sistema que usa cAER para capturar y normalizar histogramas (arriba) respecto del sistema que usa VHDL y HLS para la captura y normalización de histogramas (medio). Ambas capturas se toman durante la ejecución del juego piedra-papel-tijera. Representación del *Pipeline* al final.

figure

Por ejemplo, los sistemas de conducción autónoma requieren de hardware especializado capaz de detectar peatones, otros vehículos, señales de tráfico, obstáculos, etc, a velocidades superiores a los 30fps en algunas situaciones extremas, para garantizar la eficiencia de estos sistemas. Los algoritmos basados en CNNs realmente potentes necesitan computar inmensas cantidades de operaciones, lo que hace todavía inviable sobrepasar los límites de 30fps incluso cuando se utilizan potentes GPUs. Sin embargo, parece que el uso de múltiples CNNs de menor complejidad y más especializadas están mitigando estos problemas. Si a esto se añaden las mejoras de rendimiento de las retinas DVS, estas soluciones estarían más cerca de un producto tangible. Con este trabajo se propone usar retinas DVS como fuente de visión para aceleradores de CNN para incrementar el número de fotogramas (histogramas) procesados por segundo hasta el límite del acelerador ejecutando la CNN propuesta, en lugar de ser la cámara el cuello de botella. Se ha demostrado que el uso de las últimas tecnologías para diseño en FPGA (HLS) con relojes relativamente lentos (60MHz), permiten a un sistema hardware de *deep-learning* acelerar su rendimiento un 67% respecto a su misma versión ejecutando parte del código de preparación de la imagen en un procesador ARM a 800MHz [9]

Agradecimientos

Este trabajo es soportado por el proyecto de excelencia del gobierno de España (con soporte del fondo europeo de desarrollo regional) COFNET (TEC2016-77785-P); y por el proyecto financiado por Samsung-Electronics-Corporation NPP (P051-15/E03) coordinado por Prof. Delbruck. El trabajo de D. Ricardo Tapiador-Morales ha sido financiado por la beca de Formación de Personal Investigador de la Universidad de Sevilla.

Referencias

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[2] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *arXiv preprint arXiv:1512.03385*, 2015.

[3] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[4] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun, "Overfeat: Integrated recognition, localization and detection using convolutional networks," *arXiv preprint arXiv:1312.6229*, 2013.

[5] N. P. Jouppi *et al.*, "In-Datacenter Performance Analysis of a Tensor Processing Unit," in *Proceedings of the 44th Annual International Symposium on Computer Architecture - ISCA2017*.

[6] A. Ignatov, R. Timofte, W. Chou, K. Wang, M. Wu, T. Hartley, and L. V. Gool, "AI benchmark: Running deep neural networks on android smartphones," *CoRR*, vol. abs/1810.01109, 2018. [Online]. Available: http://arxiv.org/abs/1810.01109

[7] P. Lichtsteiner, C. Posch, and T. Delbruck, "A 128x128 120 dB 15 μs latency asynchronous temporal contrast vision sensor," *IEEE Journal of Solid-State Circuits*, vol. 43, no. 2, pp. 566–576, Feb 2008.

[8] C. Brandli, R. Berner, M. Yang, S.-C. Liu, and T. Delbruck, "A 240×180 130 dB 3 μs latency global shutter spatiotemporal vision sensor," *IEEE Journal of Solid-State Circuits*, vol. 49, no. 10, 2014.

[9] A. Aimar, H. Mostafa, E. Calabrese, A. Rios-Navarro, R. Tapiador-Morales, I. Lungu, M. B. Milde, F. Corradi, A. Linares-Barranco, S. Liu, and T. Delbruck, "Nullhop: A flexible convolutional neural network accelerator based on sparse representations of feature maps," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 30, no. 3, pp. 644–656, March 2019.

[10] Y.-H. Chen, T. Krishna, J. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," in *IEEE International Solid-State Circuits Conference*, 2016.

[11] P.-H. Pham, D. Jelaca, C. Farabet, B. Martini, Y. LeCun, and E. Culurciello, "Neuflow: Dataflow vision processing system-on-a-chip," in *2012 IEEE 55th International Midwest Symposium on Circuits and Systems*.

[12] Z. Du, R. Fasthuber, T. Chen, P. Ienne, L. Li, T. Luo, X. Feng, Y. Chen, and O. Temam, "ShiDianNao," *ISCA 2015*.

[13] J. Sim, J.-S. Park, M. Kim, D. Bae, Y. Choi, and L.-S. Kim, "14.6 A 1.42 TOPS/W deep convolutional neural network recognition processor for intelligent IoE systems," in *2016 IEEE International Solid-State Circuits Conference (ISSCC)*. IEEE, 2016, pp. 264–265.

[14] B. Moons and M. Verhelst, "A 0.3-2.6 TOPS/W precision-scalable processor for real-time large-scale ConvNets," *IEEE Symposium on VLSI Circuits, Digest of Technical Papers*, vol. 2016-Sept, pp. 1–2.

[15] B. Moons, R. Uytterhoeven, W. Dehaene, and M. Verhelst, "14.5 Envision: A 0.26-to-10TOPS/W subword-parallel dynamic-voltage-accuracy-frequency-scalable Convolutional Neural Network processor in 28nm FDSOI," in *2017 IEEE International Solid-State Circuits Conference (ISSCC)*, 2017, pp. 246–247.

[16] J. Lee, C. Kim, S. Kang, D. Shin, S. Kim, and H.-J. Yoo, "UNPU: a 50.6 TTOPS/W unified deep neural network accelerator with 1b-to-16b fully-variable weight bit-precision," in *2018 IEEE International Solid-State Circuits Conference (ISSCC)*. IEEE, feb 2018.

[17] S. Yin, P. Ouyang, S. Tang, F. Tu, X. Li, L. Liu, and S. Wei, "A 1.06-to-5.09 TOPS/W reconfigurable hybrid-neural-network processor for deep learning applications," in *2017 IEEE Symposium on VLSI Circuits*.

[18] D. Shin, J. Lee, J. Lee, and H.-J. Yoo, "14.2 DNPU: An 8.1TOPS/W reconfigurable CNN-RNN processor for general-purpose deep neural networks," in *2017 IEEE International Solid-State Circuits Conference (ISSCC)*. IEEE, feb 2017, pp. 240–241.

[19] A. Yousefzadeh, M. Jabłoński, T. Iakymchuk, A. Linares-Barranco, A. Rosado, L. A. Plana, S. Temple, T. Serrano-Gotarredona, S. B. Furber, and B. Linares-Barranco, "On multiple aer handshaking channels over high-speed bit-serial bidirectional lvds links with flow-control and clock-correction on commercial fpgas for scalable neuromorphic systems," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 11, no. 5, pp. 1133–1147, Oct 2017.

[20] A. Rios-Navarro, J. P. Dominguez-Morales, R. Tapiador-Morales, D. Gutierrez-Galan, A. Jimenez-Fernandez, and A. Linares-Barranco, "A 20mevps/32mev event-based usb framework for neuromorphic systems debugging," in *2016 Second International Conference on Event-based Control, Communication, and Signal Processing (EBCCSP)*, June 2016, pp. 1–6.

[21] A. Linares-Barranco, F. Gómez-Rodríguez, V. Villanueva, L. Longinotti, and T. Delbrück, "A usb3.0 fpga event-based filtering and tracking framework for dynamic vision sensors," in *2015 IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2015, pp. 2417–2420.

[22] R. Berner, T. Delbruck, A. Civit-Balcells, and A. Linares-Barranco, "A 5 meps $100 usb2.0 address-event monitor-sequencer interface," in *International Symposium on Circuits and Systems, (ISCAS), 2007*. IEEE, 2007.

[23] A. Rios-Navarro, R. Tapiador-Morales, A. Jimenez-Fernandez, C. Amaya, M. Dominguez-Morales, T. Delbruck, and A. Linares-Barranco, "Performance evaluation over hw/sw co-design soc memory transfers for a cnn accelerator," in *2018 IEEE 18th International Conference on Nanotechnology (IEEE-NANO)*, July 2018, pp. 1–4.

Análisis de una arquitectura segmentada para DNNs dispersas en sistemas empotrados

Adrián Alcolea*, Javier Olivito*, Javier Resano*, Hortensia Mecha†

***Resumen*—Las Redes Neuronales Profundas (DNNs) están siendo utilizadas en gran cantidad de aplicaciones, y sus altos requerimientos tanto en memoria como en computación suponen un gran reto, especialmente en el ámbito de sistemas empotrados. Técnicas como la poda generan gran dispersión en dichas redes al transformar buena parte de sus pesos en cero, lo que ofrece importantes posibilidades para su optimización mediante aceleradores hardware. Al sacar ventaja de la dispersión se logra una mayor velocidad y un menor consumo, generalmente a costa de un mayor uso de área. Nuestra propuesta consiste en una arquitectura segmentada capaz de eludir todas las operaciones innecesarias durante la inferencia en DNNs. Se ha implementado en una FPGA Xilinx UltraScale+ y se han tomado medidas reales de área y eficiencia energética. También se propone una metodología de evaluación para comparar arquitecturas densas y dispersas igualándolas en área. Obtener ventajas de la dispersión depende también de la precisión aritmética. Nuestra arquitectura resulta claramente superior con aritméticas de 32 bits, o bien cuando la dispersión es muy alta. Sin embargo, en aritméticas de 8 bits o en redes con muy poca dispersión merece más la pena desarrollar una arquitectura densa con más unidades aritméticas que implementar el soporte para redes dispersas. Consideramos que las FPGAs son el dispositivo ideal para este tipo de aceleradores, ya que permiten cargar el acelerador más indicado en tiempo de ejecución.**

***Palabras Clave*—DNN, CNN, Dispersión, Eficiencia, Sistemas empotrados, FPGA.**

I. Introducción

LAS Redes Neuronales Profundas (DNNs) han demostrado su utilidad en gran cantidad de campos. En concreto las Redes Neuronales Convolucionales (CNNs) son unas de las más utilizadas para la clasificación de imágenes.

Estas últimas son especialmente intensivas en cálculo, llegando a necesitar varios millones de operaciones en tareas como el reconocimiento de imágenes. Principalmente se entrenan sobre GPUs, aunque para la inferencia se utilizan tanto GPUs como CPUs. Sin embargo, en el caso de sistemas empotrados, estas no son una buena opción por su alto consumo de energía y recursos.

Se han dedicado muchos estudios tanto a la reducción del número de operaciones necesarias como a la cantidad de memoria que ocupan las redes. Desde pasar de aritméticas de punto flotante a punto fijo [1], [2], [3] o trabajar con pesos de menor precisión [4], [5], hasta algoritmos completos que incluyen técnicas como la poda y la compresión de filtros [6]. En concreto, la poda genera redes altamente dispersas, con lo que la cantidad de operaciones realmente necesarias es muy inferior [7], [8], [9], [10], [11].

No obstante, aprovechar estas técnicas tiene sus contrapartes. La compresión de los filtros requiere hardware extra para gestionar los índices, y para sacar partido de la dispersión es necesario localizar de antemano las operaciones útiles y gestionar accesos no uniformes a memoria. En nuestro trabajo, exploramos el equilibrio entre rendimiento, área y eficiencia energética de una arquitectura segmentada para DNNs dispersas.

Hemos diseñado un acelerador capaz de trabajar con los filtros comprimidos y realizar únicamente las operaciones útiles, limitando además los accesos de memoria a los datos necesarios para estas.

El acelerador está implementado sobre una FPGA Xilinx Zynq UltraScale+ y diseñado en lenguaje VHDL. Hemos utilizado datos sintéticos para caracterizar el comportamiento en diferentes rangos de dispersión y con diferentes tamaños de aritmética.

II. Estado del arte

Un reciente estudio de S.Mittal [12] analiza el estado del arte y las posibilidades futuras para el soporte de DNNs en ASICs y FPGAs. En él se destaca el aprovechamiento de la dispersión como una técnica muy potente para reducir la carga computacional y los requisitos de memoria de las DNNs. Otro estudio [13] plantea que existe una gran necesidad en el campo de las CNNS sobre FPGAs de soporte para redes dispersas y matrices comprimidas. Cualquier red tiene ciertos niveles de dispersión, pero las técnicas de poda son esenciales para lograr redes altamente dispersas. Estas técnicas fueron propuestas originalmente en [7], [8], y últimamente se están estudiando ampliamente y proponiendo nuevos avances [9], [10], [11]. Estas propuestas consisten en procesos iterativos en los que se identifica aquellos pesos susceptibles de ser sustituídos por ceros, para luego volver a hacer 'fine-tuning' del resto de pesos. Los modelos resultantes mantienen los niveles de precisión con muchos menos parámetros, lo que permite realizar optimizaciones basadas en esta dispersión. Sin embargo, al sacar partido de la dispersión se pierde la regularidad en los accesos de memoria, lo que puede llegar a generar incluso efectos negativos en el rendimiento, como se explica en [14].

Recientemente, diferentes trabajos han presentado arquitecturas específicas para redes dispersas. Cnvolutin fue el primer acelerador en evitar en parte algunas operaciones innecesarias [15]. Los autores propusieron una técnica para

*Grupo de Arqitectura de Computadores, Universidad de Zaragoza, e-mails: [alcolea, jolivito, jresano]@unizar.es

†Grupo de Hardware Dinámicamente Reconfigurable, Universidad Complutense de Madrid, e-mail: horten@ucm.es

Este trabajo ha sido financiado por el proyecto TIN2016-76635-C2-1-R (AEI/FEDER, UE), el Gobierno de Aragón (Grupo T58_17R) y FEDER 2014-2020 "Construyendo Europa desde Aragón".

evitar las operaciones con un cero en la activación. Han *et al.* han presentado "EIE: Efficient Inference Engine on Compressed Deep Neural Network" [16]. Este diseño trabaja con los filtros comprimidos e incluye soporte para llevar a cabo únicamente las operaciones útiles en capas 'fully-connected'. UCNN propone una técnica basada en la factorización para sustituir multiplicaciones por sumas, además de aprovechar la dispersión de los filtros [17]. Cambricon-X [18] es otro acelerados para CNNs dispersas que aprovecha la dispersión en los filtros pero no en las activaciones. NullHop [19] sin embargo, aprovecha la dispersión en las activaciones pero no en los filtros.

SCNN es un acelerador para CNNs orientado a alto rendimiento [20]. Incluye varios procesadores, cada uno de los cuales realiza una convolución en paralelo mediante productos cartesianos. Es un enfoque muy potente que permite el reuso de datos y evita las operaciones con algún cero en los operandos. Sin embargo, esta arquitectura requiere grandes buffers para guardar resultados parciales, así como grandes crossbars para unirlos con los multiplicadores. Como resultado, el soporte para aprovechar la dispersión incrementa el área de su chip en un 34%, incluso contando con una memoria de activación un 50% más pequeña que en su arquitectura densa. En cuanto al rendimiento, logran 'speedups' de 2,19 a 3,52 para varias CNNs conocidas.

Resumiendo, únicamente dos diseños evitan por completo todas las operaciones con un cero: EIE, diseñado para capas 'fully-connected', y SCNN, diseñado para capas convolucionales. Aunque nuestro diseño soporta ambas, 'fully-connected' y convolucionales, nos hemos centrado principalmente en las segundas ya que son mucho más intensivas en cálculo. SCNN también se centra en estas capas, pero está diseñado para alto rendimiento y requiere de una gran cantidad de recursos hardware, mientras que nuestro diseño está orientado a sistemas empotrados, donde los recursos hardware y el consumo energético están muy limitados. Por lo tanto, nuestro principal objetivo es la eficiencia. De hecho, nuestra arquitectura alcanza prácticamente su máximo rendimiento en la mayoría de situaciones, es decir, una operación útil por cada ciclo de reloj en cada procesador MAC, mientras que SCNN se queda muy lejos de su pico de rendimiento en las capas con mucha dispersión.

La metodología de evaluación utilizada en todos estos trabajos previos compara las mejoras en velocidad y eficiencia energética entre un diseño para redes dispersas y uno denso con la misma cantidad de recursos aritméticos. Este análisis tiene cierta utilidad, pero consideramos que es necesario un enfoque alternativo capaz de evaluar correctamente los beneficios de aprovechar la dispersión. Teniendo en cuenta que el aprovechamiento de la dispersión supone un aumento del área, nuestra propuesta es dotar a la arquitectura densa de más recursos aritméticos hasta igualar ambas superficies. Por ejemplo, los multiplicadores en SCNN suponen un 6% de su área, mientras que el soporte para gestionar la dispersión supone más del 50%.

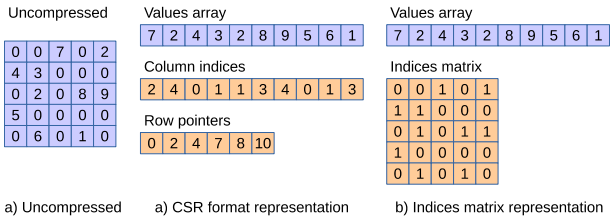


Fig. 1. Comparación entre CSR y el formato de índices.

III. Compresión de los filtros

El formato de compresión que proponemos consiste en un tensor de bits, que llamaremos índices, indicando con un uno aquellas posiciones en las que hay un valor distinto a cero en el original y un vector que contiene los valores distintos de cero. En la figura 1 se puede ver una comparativa de este formato con compressed sparse row (CSR), uno de los formatos más típicamente utilizados.

Hemos decidido utilizar este formato principalmente por dos razones: por un lado, permite la identificación de operaciones útiles de forma eficiente en términos de hardware, y por otro, mejora los ratios de compresión de un CSR.

En nuestro diseño únicamente se comprimen los filtros, ya que los recursos extra necesarios para gestionar la compresión de las activaciones no compensa, teniendo en cuenta que hay que reservar espacio para el peor caso. También es necesario generar los índices de la activación de cara a la identificación de operaciones útiles.

IV. Identificación de operaciones útiles

Para la identificación de las operaciones útiles nos apoyamos en la estructura de compresión explicada. Partiendo de los índices del filtro y la activación, basta con realizar una operación *and* bit a bit para obtener las posiciones en las que ambas tienen un dato. Esto se hace cargando secciones de ambas estructuras de índices y trabajando sobre ellas. Para ello es necesario gestionar las posiciones relativas en las que en cada momento nos estamos moviendo dentro del filtro y la activación, paralelamente con la gestión de la convolución como tal.

En la figura 2 pueden verse dos iteraciones consecutivas sobre una sección en la que aparecen dos operaciones útiles, que se corresponden con aquellas posiciones en las que tanto filtro como memoria tienen un dato distinto de cero, es decir, marcadas con un 1 en las secciones de los respectivos índices. Como puede verse, una vez encontrada la primera pareja, se enmascara esa parte de ambas secciones al pasar a la segunda iteración.

En la figura 3 puede verse un esquema de la unidad de hardware responsable de estas operaciones. Los 'offsets' devueltos son la información necesaria para posteriormente generar, junto con los índices de la convolución, las direcciones a las que acceder para buscar los datos en el filtro y la activación.

V. Arquitectura densa

El acelerador denso que hemos diseñado como base para comparar está compuesto por N unidades de procesamiento

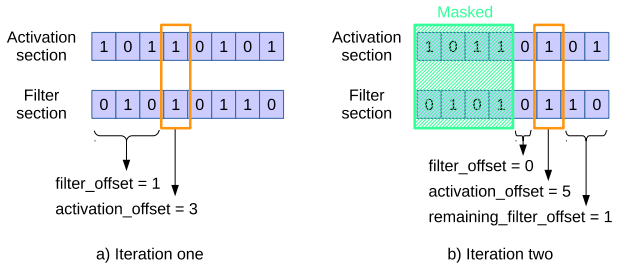


Fig. 2. Identificación de operaciones útiles.

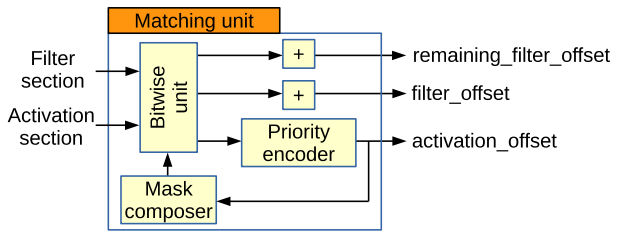


Fig. 3. Unidad responsable de la identificación de operaciones útiles.

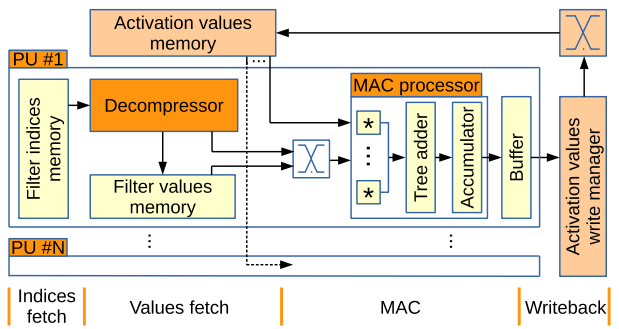


Fig. 4. Arquitectura densa.

(PUs), cada una de las cuales trabaja en paralelo con un filtro diferente, aprovechando así el paralelismo 'interfiltro'. A su vez, cada PU tiene M multiplicadores que aprovechan el paralelismo 'intrafiltro'. Además permite operar directamente sobre los filtros comprimidos. Tanto la activación de entrada como la de salida están completamente cargadas en memoria 'on-chip'. A la vez que los N primeros filtros de una capa van trabajando, se van cargando los N siguientes para ocultar la latencia de memoria. La figura 4 es una representación de los principales elementos de nuestra arquitectura densa, que como puede verse tiene cuatro etapas principales.

A. Etapa 1: Carga de los índices

Se cargan los M índices correspondientes a los operandos de las M multiplicaciones que se realizarán en la tercera etapa.

B. Etapa 2: Carga de los valores

Se cargan globalmente los M valores de la activación y, localmente, los M valores correspondientes de cada filtro. Las direcciones de estos últimos se calculan a partir de los índices de la etapa anterior.

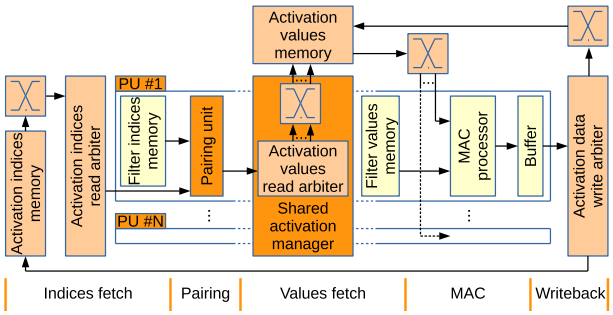


Fig. 5. Arquitectura dispersa.

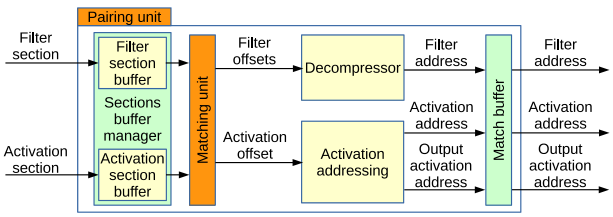


Fig. 6. Unidad de Emparejamiento.

C. Etapa 3: MAC

Una vez cargados los valores se realizan los M productos y sus valores se acumulan en un solo ciclo mediante un sumador en árbol.

D. Etapa 4: Escritura

Una vez terminado un paso de la convolución, el valor final se guarda en la activación de salida. Puesto que esta memoria es compartida, son necesarios un buffer intermedio y un árbitro para evitar colisiones en escritura.

VI. Arquitectura dispersa

En la figura 5 se pueden observar las cinco etapas de nuestra arquitectura dispersa.

A. Etapa 1: Carga de los índices

En este caso es necesario cargar índices tanto de los filtros como de la activación. Aunque únicamente los primeros están comprimidos, ambos índices se utilizarán en la etapa siguiente para el emparejamiento. Puesto que la memoria de activación es compartida y en este caso los filtros no avanzan a la vez, es necesario un arbitraje y aumentar el número de bancos para evitar conflictos de lectura.

B. Etapa 2: Emparejamiento

En la figura 6 se puede ver un esquema más detallado de la unidad de emparejamiento. En ella se cruza la información de los índices para encontrar las posiciones en las que se encuentran datos para las operaciones útiles, es decir, aquellas que no tienen ningún cero.

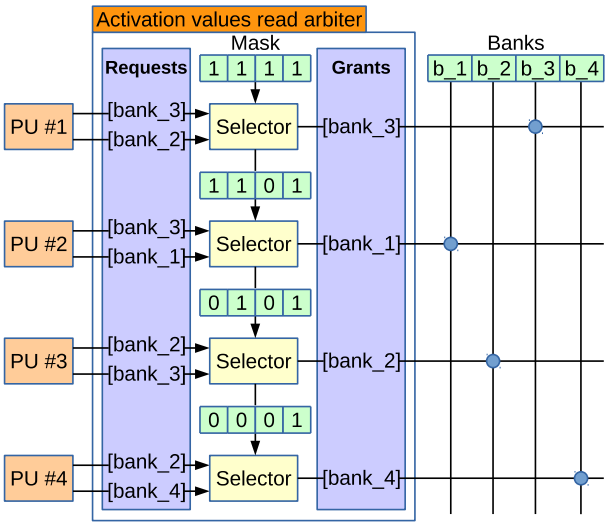


Fig. 7. Activation values read arbiter workflow.

C. Etapa 3: Carga de los valores

En una arquitectura dispersa, los accesos a la memoria de activación dejan de ser regulares, con lo que es necesario arbitrar y proporcionar más recursos si queremos evitar paradas por conflictos de acceso. Para ello, nuestro árbitro es capaz de gestionar varias peticiones desde cada PU para, en caso de conflicto en un banco, tener la posibilidad de ofrecer datos de otra petición de la misma PU.

La figura 7 muestra un ejemplo reducido del funcionamiento del árbitro. Las PUs tienen prioridades fijas, de modo que las primeras tienen mayor prioridad que las siguientes. Una vez seleccionado un banco, este se enmascara de forma que las siguientes PUs ya no pueden solicitar datos del mismo. En nuestro diseño hemos encontrado que teniendo el doble de bancos que de PUs y estableciendo un límite de cuatro peticiones por PU logramos que apenas haya paradas por un conflicto en memoria.

D. Etapa 4: MAC

Una vez cargados los operandos se realizan las multiplicaciones y se mantiene el resultado acumulado de cada PU.

E. Etapa 5: Escritura

Ocurre lo mismo que en la etapa correspondiente de la arquitectura densa.

VII. Resultados experimentales

Ambos diseños, el denso y el disperso, han sido implementados en la FPGA de la placa de evaluación Xilinx Zynq UltraScale+ ZCU104 [21]. La CPU se ha utilizado para gestionar el envío de los datos a las memorias principales. Las medidas de consumo se han realizado con un medidor de potencia digital Yokogawa WT210 [22]. Se han tomado medidas dinámicas, es decir, tomando la diferencia de consumo entre el diseño parado en 'idle' y el diseño corriendo.

El objetivo de nuestras mediciones es caracterizar el comportamiento de ambos diseños en términos de rendimiento, recursos hardware y eficiencia energética, en función del porcentaje de operaciones útiles y el tamaño de la aritmética. Para ello hemos desarrollado una colección de datos variando el ratio de operaciones útiles de 0 a 1 y los hemos lanzado con aritméticas de 8, 16 y 32 bits. Los resultados de las figuras están normalizados a la base de referencia de cada caso de estudio.

La tabla I muestra los detalles de diseño, recursos hardware, frecuencia máxima y consumo energético de cada una de las implementaciones. Hay tres grupos de diseños utilizados para las pruebas:

a) Dispersa: arquitectura dispersa con un multiplicador por PU.
b) Densa base: arquitectura densa con un multiplicador por PU.
c) Densa equivalente: arquitectura densa donde el número de multiplicadores por PU se ha definido para aproximarse lo más posible en área a la arquitectura dispersa equivalente.

En primer lugar hemos comparado las implementaciones *a* y *b*. La figura 8a muestra la ganancia en rendimiento en función del porcentaje de operaciones útiles, mientras que la figura 8b muestra la ganancia en términos de consumo energético. En el segundo caso, la ganancia varía también en función de la aritmética. También puede verse en las barras superiores el incremento en el uso de recursos hardware generado por la lógica necesaria para gestionar la dispersión.

En escenarios con mucha dispersión las ganancias son muy grandes, pero hay que remarcar que el diseño disperso está utilizando desde un 50% más hasta casi un 200% más de recursos que el diseño denso. Por eso nos preguntamos: ¿cuál sería la ganancia real si ambos diseños utilizaran la misma cantidad de recursos hardware? Para ello hemos diseñado la implementación *c*, con lo que podemos realizar comparaciones mucho más justas frente a la implementación *a* que nos permitan decidir cuándo merece realmente la pena desarrollar o no soporte específico para aprovechar la dispersión.

Las figuras 9a y 9b muestran los resultados de comparar las implementaciones *a* y *c*. como podemos ver, ahora la ganancia no siempre es tan grande. De hecho, para diseños con una aritmética de 8 bits solo merece la pena cuando hay menos de un 10% de operaciones útiles, para aritméticas de 16 bits la ganancia comienza de un 25% para abajo, mientras que con 32 bits con menos de un 50% de operaciones útiles ya se mejora el rendimiento. En cuanto a la eficiencia energética las cifras son más favorables al modelo disperso.

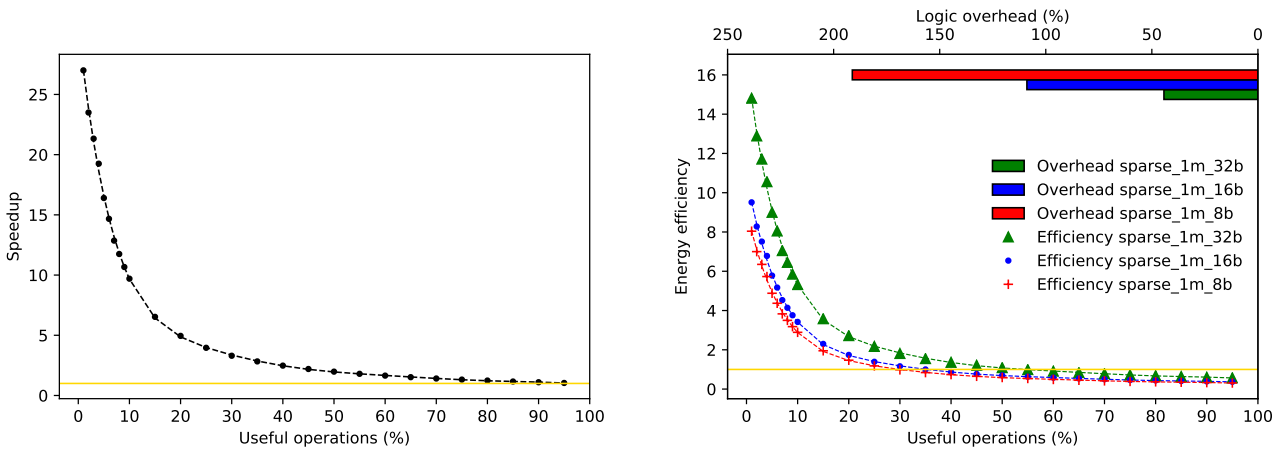
Uno de los objetivos de nuestra arquitectura dispersa consiste en maximizar la utilización de recursos aritméticos. Como se puede ver en la figura 10, incluso para redes con muy pocas operaciones útiles, el uso de los multiplicadores sigue siendo prácticamente del 100%.

VIII. Conclusión

Hemos propuesto una arquitectura dispersa para sistemas empotrados que maximiza el rendimiento y reduce el consumo

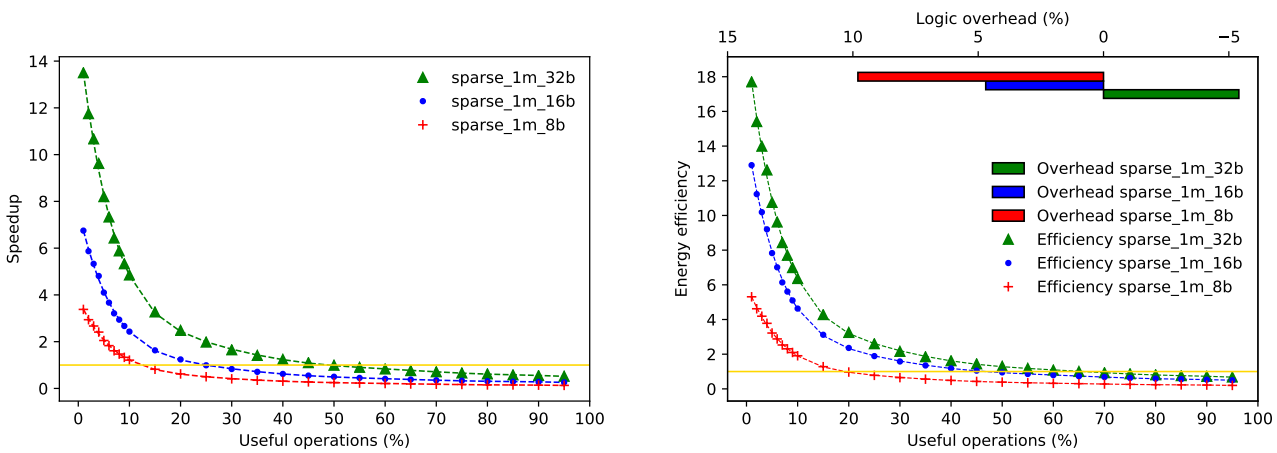
TABLA I
IMPLEMENTACIONES PARA LAS PRUEBAS

Implementación	Aritmética (bits)	PUs	Mults / PU	Lógica (LUT %)	Frec. Max. (MHz)	Consumo (mW)
sparse_1m_8b	8			5.94	124	131
sparse_1m_16b	16	8	1	7.94	110	247
sparse_1m_32b	32			12.32	106	842
dense_1m_8b	8			2.04	123	39
dense_1m_16b	16	8	1	3.80	112	87
dense_1m_32b	32			8.54	101	462
dense_8m_8b	8		8	5.41	123	206
dense_4m_16b	16	8	4	7.58	115	472
dense_2m_32b	32		2	13.02	101	1104



(a) Aumento de Velocidad (b) Eficiencia Energética

Fig. 8. Aumento de Velocidad y Eficiencia Energética de las implementaciones dispersas normalizadas a la densa base.



(a) Aumento de Velocidad (b) Eficiencia Energética

Fig. 9. Aumento de Velocidad y Eficiencia Energética de las implementaciones dispersas normalizadas a la densa equivalente.

energético manteniendo el uso de recursos hardware. Nuestro diseño evita todas las operaciones con algún cero en los operandos y mantiene prácticamente al máximo el uso de los recursos aritméticos incluso con muy alta dispersión.

Hemos evaluado el rendimiento y la eficiencia energética del diseño disperso y el denso con diferentes rangos de dispersión y diferentes aritméticas. Las comparaciones que normalmente se realizan entre un diseño disperso y su equivalente denso no nos parecen justas, puesto que el primero utiliza más recursos hardware. Por lo tanto, hemos realizado comparaciones con un diseño denso adaptable en área, de forma que puede aprovechar mayor paralelismo, de forma que nos ha sido posible caracterizar en qué escenarios merece realmente la pena desarrollar un diseño disperso y en cuales merece más la pena dotar de recursos al denso.

La propia dispersión de la red es uno de los parámetros a tener en cuenta, pero también la aritmética, puesto que el coste en hardware de los multiplicadores no escala linealmente.

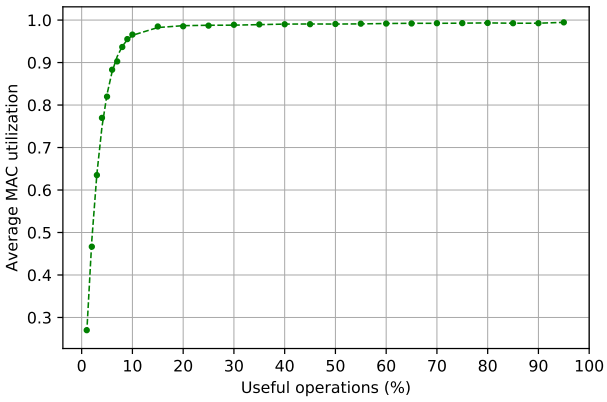


Fig. 10. Uso medio de los multiplicadores.

Según nuestros resultados, con aritméticas de 32 bits los beneficios de sacar ventaja de la dispersión son claros, mientras que en aritméticas de 8 bits es difícil sacar mayor ventaja que la obtenida incrementando el paralelismo con más multiplicadores. Actualmente, diferentes aceleradores presentados trabajan con aritméticas de 16 bits. Para estas últimas, hemos logrado sacar beneficio en términos de eficiencia energética cuando menos del 50% son operaciones útiles, mientras que los beneficios en términos de rendimiento comienzan por debajo del 25% de operaciones útiles.

Puesto que no en todos los escenarios conviene utilizar un acelerador disperso, consideramos que los dispositivos reprogramables como las FPGAs son más útiles en muchos contextos que los ASICs, ya que permiten adaptarse fácilmente al contexto, cargando el acelerador adecuado a cada caso.

Dada la cantidad de investigaciones en torno a las técnicas de poda, es de esperar que la cantidad de modelos altamente dispersos no haga sino aumentar, con lo que consideramos el soporte para dispersión una técnica tremendamente necesaria para sacar ventaja de estas oportunidades de optimización.

Referencias

[1] P. Gysel, M. Motamedi, and S. Ghiasi, "Hardware-oriented approximation of convolutional neural networks," *Computing Research Repository*, vol. eprint arXiv:1604.03168, 2016. [Online]. Available: http://arxiv.org/abs/1604.03168

[2] M. B. Milde *et al.*, "Adaption: Toolbox and benchmark for training convolutional neural networks with reduced numerical precision weights and activation," *Computing Research Repository*, vol. eprint arXiv:1711.04713, 2017. [Online]. Available: http://arxiv.org/abs/1711.04713

[3] V. Sze *et al.*, "Efficient processing of deep neural networks: A tutorial and survey," *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, December 2017.

[4] C. Zhu *et al.*, "Trained ternary quantization," *Computing Research Repository*, vol. eprint arXiv:1612.01064, 2017. [Online]. Available: http://arxiv.org/abs/1612.01064

[5] A. Zhou *et al.*, "Incremental network quantization: Towards lossless cnns with low-precision weights," *Computing Research Repository*, vol. eprint arXiv:1702.03044, 2017. [Online]. Available: http://arxiv.org/abs/1702.03044

[6] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *Computing Research Repository*, vol. eprint arXiv:1510.00149, 2015. [Online]. Available: http://arxiv.org/abs/1510.00149v5

[7] Y. LeCun, J. S. Denker, and S. A. Solla, "Optimal brain damage," in *Advances in Neural Information Processing Systems 2*, D. S. Touretzky, Ed. Morgan-Kaufmann, 1990, pp. 598–605. [Online]. Available: http://papers.nips.cc/paper/250-optimal-brain-damage.pdf

[8] B. Hassibi and D. G. Stork, "Second order derivatives for network pruning: Optimal brain surgeon," in *Advances in Neural Information Processing Systems 5*, S. J. Hanson, J. D. Cowan, and C. L. Giles, Eds. Morgan-Kaufmann, 1993, pp. 164–171. [Online]. Available: http://papers.nips.cc/paper/647-second-order-derivatives-for-network-pruning-optimal-brain-surgeon.pdf

[9] S. Han *et al.*, "Learning both weights and connections for efficient neural network," in *Advances in Neural Information Processing Systems 28*, C. Cortes *et al.*, Eds. Curran Associates, Inc., 2015, pp. 1135–1143. [Online]. Available: http://papers.nips.cc/paper/5784-learning-both-weights-and-connections-for-efficient-neural-network.pdf

[10] T.-J. Yang, Y.-H. Chen, and V. Sze, "Designing energy-efficient convolutional neural networks using energy-aware pruning," *Computing Research Repository*, vol. eprint arXiv:1611.05128, 2016. [Online]. Available: https://arxiv.org/pdf/1611.05128.pdf

[11] M. Zhu and S. Gupta, "To prune, or not to prune: exploring the efficacy of pruning for model compression," *Computing Research Repository*, vol. eprint arXiv:1710.01878, 2017. [Online]. Available: https://arxiv.org/pdf/1710.01878.pdf

[12] S. Mittal, "A survey of fpga-based accelerators for convolutional neural networks," *Neural Computing and Applications*, 09 2018.

[13] S. I. Venieris, A. Kouris, and C.-S. Bouganis, "Toolflows for mapping convolutional neural networks on fpgas: A survey and future directions," *ACM Computing Surveys*, vol. 51, no. 3, pp. 56:1–56:39, June 2018. [Online]. Available: http://doi.acm.org/10.1145/3186332

[14] J. Yu *et al.*, "Scalpel: Customizing dnn pruning to the underlying hardware parallelism," in *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, 2017, pp. 548–560. [Online]. Available: https://ieeexplore.ieee.org/document/8192500

[15] J. Albericio *et al.*, "Cnvlutin: Ineffectual-neuron-free deep neural network computing," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, June 2016, pp. 1–13.

[16] S. Han *et al.*, "EIE: Efficient inference engine on compressed deep neural network," *Computing Research Repository*, vol. eprint arXiv:1602.01528, 2016. [Online]. Available: http://arxiv.org/abs/1602.01528

[17] K. Hegde *et al.*, "Ucnn: Exploiting computational reuse in deep neural networks via weight repetition," *Computing Research Repository*, vol. eprint arXiv:1804.06508, 2018. [Online]. Available: https://arxiv.org/pdf/1804.06508.pdf

[18] S. Zhang *et al.*, "Cambricon-x: An accelerator for sparse neural networks," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Oct 2016, pp. 1–12. [Online]. Available: https://ieeexplore.ieee.org/document/7783723

[19] A. Aimar *et al.*, "Nullhop: A flexible convolutional neural network accelerator based on sparse representations of feature maps," *Computing Research Repository*, vol. eprint arXiv:1706.01406, 2018. [Online]. Available: https://arxiv.org/pdf/1706.01406.pdf

[20] A. Parashar *et al.*, "Scnn: An accelerator for compressed-sparse convolutional neural networks," *Computing Research Repository*, vol. eprint arXiv:1708.04485, 2017. [Online]. Available: http://arxiv.org/abs/1708.04485

[21] Xilinx, "Zynq ultrascale+ mpsoc zcu104 evaluation kit, soc and fpga platform," https://www.xilinx.com/products/boards-and-kits/zcu104.html.

[22] Yokogawa, "Wt210/wt230 digital power meters," http://tmi.yokogawa.com/products/digital-power-analyzers/digital-power-analyzers/wt210wt230-digital-powermeters/#tm-wt210_01.htm.

Modelado y caracterización del flujo de tráfico en entornos urbanos

Jorge Luis Zambrano-Martinez[1], Carlos T. Calafate[1] , David Soler[2] , Juan-Carlos Cano[1]y Pietro Manzoni[1]

Resumen— **En la actualidad, uno de los principales desafíos a los que se enfrentan las grandes áreas metropolitanas es la congestión del tráfico. Para solucionar este problema, un control de tráfico adecuado podría producir muchos beneficios, incluida la reducción de las emisiones de contaminantes y de los tiempos de viaje. Si fuera posible caracterizar el estado del tráfico mediante la predicción de las condiciones futuras del tráfico, se podrá optimizar la ruta de vehículos autónomos. Además, si se pudieran tomar estas medidas para mitigar de manera preventiva los efectos de la congestión y problemas relacionados, se podría mejorar el flujo de tráfico en general. Este artículo realiza un estudio experimental de la distribución del tráfico en la ciudad de Valencia, España, que caracteriza las diferentes calles de la ciudad en términos de carga de vehículos con respecto al tiempo de viaje en condiciones de tráfico en hora punta. Resultados experimentales basados en trazas de tráfico vehicular realistas de la ciudad de Valencia muestran que solo algunos segmentos de calle se encuentran dentro de la teoría general del flujo vehicular, ofreciendo un buen ajuste con regresión cuadrática, mientras que un gran número de segmentos de calles se ubican en otras categorías. Aunque en algunos casos tales discrepancias están relacionadas con la escasez de tráfico por esas zonas, inyectar vehículos adicionales muestra que aún persisten desajustes importantes. Por lo tanto, en este artículo proponemos una ecuación que pertenece a la familia sigmoide para caracterizar los tiempos de viaje en un segmento; concretamente, aplicamos una regresión logística, mediante la cual ha sido posible mejorar significativamente los resultados del ajuste de curvas para la mayoría de los segmentos de calles analizados. En base a nuestros resultados de regresión, realizamos un análisis de clustering de los diferentes segmentos de calles, el cual demuestra que éstos se pueden clasificar en tres categorías bien definidas, evidenciando una distribución de tráfico predecible utilizando la regresión logística en toda la ciudad durante las horas punta, y que permitie optimizar el tráfico para vehículos autónomos.**

Palabras clave— **Vehículos Autónomos; Predicción del Tráfico; Comportamiento del Tráfico; SUMO; DFROUTER; Tráfico Urbano; Valencia**

I. Introducción

Uno de los problemas más críticos para las autoridades de las ciudades es el aumento de las emisiones de dióxido de carbono (CO_2) causadas por la congestión del tráfico, ya que los atascos y los elevados tiempos de trayecto de los vehículos se asocian con un uso ineficiente del combustible [1]. A medida que avanzamos gradualmente hacia un nuevo paradigma centrado en los vehículos automatizados, somos capaces de dotar a los administradores de tráfico con formas más sofisticadas de regular el tráfico. Las estrategias para abordar este problema se basan en las regulaciones de tiempo del semáforo. Por ejemplo, PDDL+ [2] es un enfoque de planificación para el control del tráfico urbano que reduce de manera eficiente la congestión de carreteras específicas. Otra solución es el sistema de Inteligencia Artificial propuesto por Pozanco et al. [3] que se basa en una estrategia de planificación automatizada declarativa para generar planes de control cuando el comportamiento predeterminado debe ser anulado. Otra estrategia es el despliegue de agentes de tráfico [4], los cuales son controlados de forma independiente por un agente de programación en línea que puede optimizar el movimiento y el control del tráfico en las intersecciones utilizando el programa generado. Entre estas nuevas técnicas de gestió}n del tráfico, la administración centralizada de rutas surge como una solución óptima que ofrece a las autoridades el control total del flujo de tráfico [5], permitiendo así que la optimización del tráfico alcance niveles máximos de efectividad al determinar la ruta que debe seguir cada vehículo. Al obtener *a priori* conocimiento del estado de congestión del tráfico, es posible optimizar la ruta de los vehículos nuevos, especialmente en el caso de los vehículos autónomos, que pueden contar con un servidor central de rutas de confianza para indicar que proporcione la ruta más adecuada a cada vehículo.

Para avanzar hacia estos nuevos paradigmas de gestión de tráfico, se hace obligatorio conocer completamente el comportamiento de los diferentes segmentos de calles de una ciudad en términos de cómo el tiempo de viaje puede variar dependiendo de la cantidad de vehículos que viajan simultáneamente en un segmento específico. En este trabajo, nos basamos en modelos de tráfico realistas que describen el comportamiento del tráfico en la ciudad de Valencia durante las horas punta, como se detalla en nuestro trabajo anterior [6]. Concretamente, a partir de las mediciones de anillos de inducción disponibles por el Ayuntamiento de Valencia [7], y mediante el uso de la herramienta DFROUTER [8], junto con una heurística que refina iterativamente el resultado producido por DFROUTER, generamos una matriz Origen-Destino (O-D) de tráfico que se asemeja a la distribución de tráfico real. De esta manera, demostramos los grados de congestión esperados, y también el impacto de ciertos eventos puntuales que generan tráfico adicional en la ciudad [9]. En este trabajo, nos centraremos en analizar, modelar y caracterizar la forma en la cual el tráfico se distribuye a lo largo de

[1]Departamento de Informática de Sistemas y Computadores (DISCA), Universitat Politècnica de València, España, e-mail:jorzamma@doctor.upv.es, calafate@disca.upv.es,{jucano, pmanzoni}@disca.upv.es
[2]Instituto de Matemática Multidisciplinar (IMM). Universitat Politècnica de València, España, e-mail: dsoler@mat.upv.es.

una ciudad, reuniendo detalles sobre la cantidad de vehículos que viajan a lo largo de los diferentes segmentos de calle, así como sus tiempos de viaje [10], y utilizando el tráfico de referencia para la ciudad de Valencia como datos de entrada para la herramienta de simulación de movilidad urbana (SUMO) [11]. En particular, muestreamos los tiempos de viaje de los vehículos que ingresan a un segmento, junto con la cantidad de vehículos que ya están en ese segmento, lo cual permite extraer una relación entre ocupación y tiempo de trayecto por segmento (tramo de calle). Mediante un análisis de regresión, mostramos que un ajuste cuadrático, a pesar de coincidir con la teoría de flujo de tráfico [12], no es adecuado en muchos casos. Por lo tanto, buscamos una función alternativa capaz de proporcionar un ajuste adecuado para la mayoría de los comportamientos de los segmentos. De esta manera, la regresión logística emerge como la solución más adecuada, ofreciendo mejoras claras en la categorización de segmentos para cualquier escenario. Además, al realizar un análisis de clustering, pudimos identificar claramente tres categorías independientes, cuyas características se discuten en este trabajo.

El resto de este documento está organizado como sigue: la Sección II presenta algunos trabajos relacionados que predicen el comportamiento del tráfico utilizando diferentes enfoques. La Sección III proporciona información sobre las herramientas SUMO y DFROUTER, junto con nuestra heurística iterativa. La Sección IV describe la metodología que se ha utilizado para lograr el modelado de tráfico por segmento a partir de datos de los anillos de inducción, y que termina en una clasificación de los diferentes segmentos de calle. En la Sección V proponemos una ecuación de la familia sigmoidal para predecir el comportamiento de los segmentos de calle que mejor se ajustan a una regresión logística. En la Sección VI analizamos el comportamiento de la congestión del tráfico a través de la regresión logística en la ciudad de Valencia, y en un área específica de la misma ciudad, incluidos los resultados del clustering realizado. Finalmente, la Sección VII concluye el documento y analiza la relevancia de los resultados obtenidos, junto con su potencial para los sistemas de gestión de tráfico del futuro.

II. Trabajos Relacionados

En las últimas décadas, muchos investigadores han desarrollado una amplia gama de modelos predictivos de tráfico vehicular desde diferentes perspectivas. Zhang y Rice [13] propusieron un método para predecir los tiempos de viaje mediante una regresión lineal, asumiendo una combinación lineal de covariables, y teniendo una estructura simple. Por otro lado, Guo et al. [14] propusieron utilizar un filtro de Kalman adaptativo para predecir el tráfico. Utilizaron métodos de series de tiempo para conocer el valor futuro de diferentes parámetros de tráfico, y para actualizar las variables estables seleccionadas. Van Hinsbergen et al. [15] propusieron utilizar un filtro de Kalman extendido localizado para resolver el problema de la lentitud que presentan estos filtros en redes grandes. Además, utilizan filtros locales para corregir el estado de los detectores de proximidad, cuyas estimaciones se basan en datos de sensores locales, para predecir el estado del tráfico.

Algunos trabajos que se ocupan de la predicción de tráfico involucran algoritmos de aprendizaje como lógica difusa [16], aunque enfrentan algunos problemas, como su baja precisión y eficiencia. Por ejemplo, Onieva et al. [17] presentaron un caso de estudio en el que un vehículo automatizado debe cooperar con un conductor para lograr maniobras sin riesgo, y desarrolló un sistema jerárquico de tres capas basado en reglas. La primera capa detecta el tipo de maniobra que se necesita, la segunda capa es la velocidad adecuada para cruzar una intersección, y la tercera capa es la velocidad real del vehículo. Hodge et al. [18] presentaron un algoritmo de red neuronal binaria para la predicción del flujo de tráfico a corto plazo utilizando datos univariados y multivariados de un solo sensor de tráfico con retrasos temporales, y combinando información de múltiples sensores de tráfico con la predicción de series de tiempo o retrasos espacio-temporales. Habtie et al. [19] presentaron un enfoque para estimar el estado del tráfico rodado utilizando la red celular existente como fuente de datos de tráfico, y utilizando un modelo para estimar el estado de la red neuronal. Porikli y Li [20] entrenaron un Modelo de Markov oculto correspondiente a cinco patrones de tráfico (parada, congestión intensa, flujo abierto, moderada y congestión vacía), y utilizaron un criterio de probabilidad máxima para determinar el estado de los modelos de Markov ocultos por separado. A diferencia de trabajos anteriores, Kunt et al. [21] se centraron en predecir la gravedad de los accidentes de tráfico en las autopistas mediante el empleo de doce parámetros relacionados con el accidente como entrada a una red neuronal artificial, combinado con un algoritmo genético y una búsqueda de patrones.

Los trabajos anteriores han desarrollado una amplia variedad de modelos de predicción de tráfico desde diferentes perspectivas, basados en métodos estadísticos o inteligencia computacional. Sin embargo, tienen inconvenientes ya que estos modelos se desarrollan con datos sintéticos, suponiendo ciertas condiciones de tráfico, y sin un flujo de tráfico realista.

En este documento proponemos una función logística para caracterizar correctamente el comportamiento del tiempo de viaje de diferentes calles en función del número medido de vehículos que se avecinan, utilizando como datos reales de tráfico como entrada. Además, clasificamos el comportamiento del tráfico vehicular para los diferentes segmentos de calle de la ciudad de Valencia a través de un algoritmo de clustering, y utilizamos una técnica estadística para describir correctamente el conjunto de datos de clustering.

III. Herramientas de simulación utilizadas

En esta sección, se presentan detalles sobre el simulador de tráfico SUMO. Del mismo modo, presentaremos una herramienta denominada DFROUTER, y explicaremos brevemente su funcionamiento de cara a generar una matriz de tráfico detallando los oríŋgenes y destinos, comúnmente conocida como matriz O-D, para la herramienta SUMO a partir de los datos de los anillos de inducción.

A. SUMO

Por lo general, el modelado del tráfico consiste en obtener algunas variables, como la hora de salida, la ruta seguida por los diferentes vehículos y, en algunos casos, la duración de la ruta. Hay que tener en cuenta que esta última no se puede calcular de una manera realista, ya que asume un determinado vehículo, conductor y estado de congestión de tráfico a lo largo de la ruta. SUMO aborda este desafío a través del modelado microscópunta detallado de ciudades y vehículos, ofreciendo diferentes paquetes para la simulación de tráfico que se han convertido en una gama completa de herramientas. De hecho, al ser un simulador de código abierto, se está mejorando constantemente y es muy aceptado por la comunidad científica. Sus características incluyen: simulación de tráfico multimodal, horarios de tráfico, soporte para varios formatos de mapas como OpenStreetMaps, capacidad para importar mapas de redes de carreteras en múltiples formatos y generar rutas desde múltiples fuentes. También ofrece soporte de simulación de alto rendimiento a través del interfaz TraCI, lo que permite realizar simulaciones interactivas cuando se combina con otro simulador (por ejemplo, OMNeT++ [24]). Una ventaja clave de SUMO es su soporte para realizar simulación multimodal, ya que no solo incluye el movimiento de vehículos en una ciudad, sino que también incluye el sistema de transporte público, redes ferroviarias, e incluso caminos peatonales. Esto significa que una modalidad de tráfico puede describirse mediante varias rutas, que pueden estar compuestas por varias subrutas. Dado que el flujo de tráfico se simula microscópicamente, cada vehículo en movimiento dentro de la red se modela individualmente, se ubica en una posición específica, y se asocia a una velocidad específica. Cada paso de tiempo tiene una duración de un segundo simulado por defecto, lo que permite la simulación discreta de la movilidad continua en el espacio. Además, las simulaciones consideran los diferentes atributos de las carreteras, como la velocidad máxima o las reglas de prioridad hacia el lado derecho, al mismo tiempo que cuentan modelos realistas del conductor.

B. Matriz O-D generada con DFROUTER

Uno de los paquetes incluidos en la versión 0.32.0 del simulador SUMO es la herramienta DFROUTER. Esta herramienta ha sido diseñada para los escenarios de carreteras basándose en la idea principal de que las carreteras están equipadas con anillos de inducción que permiten medir el flujo de entrada y salida de las mismas. En base a esta información, DFROUTER permite reconstruir el número de vehículos y rutas que se inyectarán en el simulador de la red de carreteras, basándose en los datos de los anillos de inducción, como el número de vehículos, flujos y velocidades, para lograr la matriz de tráfico O-D deseada. En otras palabras, esta herramienta, a partir del recuento realizado por los anillos de inducción para las diferentes calles de una ciudad, permite estimar las posibles rutas de vehículos que coinciden con dicha entrada. En un trabajo anterior [9], utilizamos datos de los anillos de inducción proporcionados por el Ayuntamiento de Valencia, España, como entrada para DFROUTER. Estos conjuntos de datos fueron generados por 520 detectores de bucle de inducción desplegados en toda la ciudad, y los valores corresponden a la hora punta (entre las 8:00 y las 9:00 a.m.) de un lunes típunta. En ese trabajo, se demostró una discrepancia significativa entre el tráfico generado y los datos originales. A continuación introdujimos una heurística iterativa que compensa este error refinando sucesivamente la salida provista por esta herramienta para lograr una matriz O-D que se asemeja a la distribución de tráfico real. La Figura 1 muestra que, como resultado, la salida de DFROUTER que utiliza nuestro proceso heurístico iterativo logra un alto nivel de coincidencia con los datos de referencia, resultando en un error inferior al 0,0001 % y, por lo tanto, es significativamente mejor que la salida inicial del DFROUTER [9].

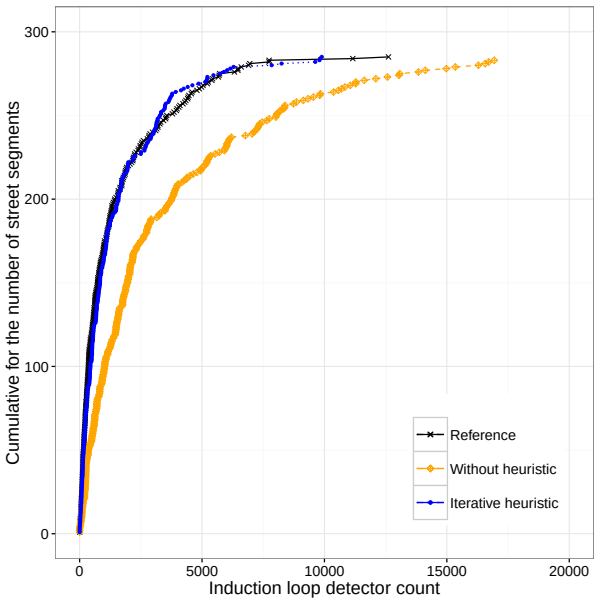


Fig. 1: Modelado de flujos de tráfico para la ciudad de Valencia. Resultados con y sin nuestra heurística iterativa .

IV. Metodología

En esta sección describimos el procedimiento seguido para caracterizar el tráfico en la ciudad de Valencia, España, desde una perspectiva microscópica, y partiendo de los diseños de carreteras de OpenStreetMap. Concretamente, nuestro objetivo es carac-

terizar segmentos de calles individuales en términos de los tiempos de viaje promedio experimentados por los vehículos para diferentes grados de congestión, siendo este último estimado en función de la cantidad de vehículos que el vehículo encuentra adelante cuando acaba de ingresar a un segmento. Para lograr este objetivo, describimos la metodología seguida para caracterizar y predecir el tráfico para los diferentes segmentos de calle. La metodología propuesta es la siguiente: primero, obtenemos los segmentos necesarios de la ciudad; luego, podemos predecir el número de vehículos en cada segmento; finalmente, caracterizamos los diferentes segmentos de calle a través de un análisis de regresión, que se complementa con un agrupamiento de estos resultados para lograr una clasificación adecuada. A continuación, detallamos nuestra propuesta para predecir el tiempo de trayecto por segmento, así como para caracterizar los diferentes segmentos de acuerdo con los tiempos de viaje del vehículo.

A. Predicción del tiempo de viaje por segmento

En esta sección, procedemos a predecir el tiempo de viaje asociado a cada segmento para diferentes grados de congestión, siendo medido como el número de vehículos ubicados en el segmento justo antes de que un nuevo vehículo ingrese al segmento (ν_n). Para lograr esta predicción, debemos tener en cuenta el tiempo de entrada (t^{ν}_{in}) y el tiempo de salida (t^{ν}_{out}) del vehículo en el segmento, así como la cantidad de carriles del segmento (l_n) por donde viaja el vehículo. Los tiempos de entrada (t_{in}) para los vehículos que ingresan en un segmento en el carril l_n están registrados en la matriz (l_n,t_{in}), mientras que los tiempos de salida (t_{out}) para los vehículos que abandonan el segmento en la línea l_n se registran en la matriz (l_n,t_{out}).

La cantidad de vehículos en un segmento antes de que un vehículo se una al mismo se incrementará siempre que t^{ν}_{in} sea menor que t^{ν}_{out}, y si ambos se refieren al mismo carril. A partir de estos datos, se obtiene el tiempo de viaje de cada vehículo en el segmento (Δt).

Como paso final, el Algoritmo 1 permite obtener el promedio de los tiempos de viaje ($\bar{\Delta t}$) asociado a diferentes grados de congestión (número de vehículos en el segmento antes de que un vehículo nuevo entre en ese segmento), junto con la cantidad de vehículos en el segmento (ν).

Algoritmo 1 Extracción de tiempos de viaje vs. muestras de carga.

Entrada: Archivo de segmento reunificado, archivos de información de segmento
Salida: Aprendizaje estadístico por archivos de segmento

para segmento **en** Archivo de segmento reunificado **hacer**
  $segmentConnected[]$ ← vector que almacena todos los bordes id conectados
  para s=0 **hasta** tamaño($segmentConnected$) **hacer**
    $segment\_info$ ← Leer lineas $segmentConnected[s]$ en archivos de información de segmento
    $segmentSorted[][]$ ← ordenar_por_t_{in}($segment\_info$)
    para t_{in}=0 **hasta** tamaño($segmentSorted$) **hacer**
      ν_n ← Número de vehículos por segmento en cada carril
      para $t_{out} = t_{in}$ **hasta** t_{out} >= 0 **paso** -1 **hacer**
        si ($segmentSorted[t_{in}][l_n]$ = $segmentSorted[t_{out}][l_n]$) **y** ($segmentSorted[t_{in}][t^{\nu}_{in}]$ <= $segmentSorted[t_{out}][t^{\nu}_{out}]$) **entonces**
          $\nu_n = \nu_n + 1$
        fin si
      fin para
    fin para
    $\bar{\Delta t}$ ← promedio de los tiempos de viaje
    ν ← cantidad de vehículos en el segmento antes de que un nuevo vehículo ingrese al segmento
  fin para
fin para

V. Predictor propuesto para los tiempos de trayecto

En un trabajo previo [10] analizamos el comportamiento del tráfico en una ciudad siguiendo los criterios de la teoría general del tráfico, observando que los resultados del ajuste de regresión polinómica pueden considerarse inadecuados para la mayoría de los segmentos.En este trabajo, nuestro objetivo es encontrar una función matemática alternativa que se adapte mejor al comportamiento que caracteriza a los diferentes segmentos de la ciudad. En este artículo, proponemos utilizar una función matemática que pertenece a la familia logística de funciones. Concretamente, elegimos la función sigmoide para representar los patrones de crecimiento detectados en nuestro conjunto de datos. Por lo tanto, por una regresión logística para predecir el resultado de una variable que puede adoptar un número limitado de categorías basadas en variables independientes o predictoras, y este tipo de regresión se usa para modelar la probabilidad de un evento que ocurre como una función de otra [25]. Con este fin, nos basamos en la función sigmoide simple definida por la siguiente expresión matemática:

$$f(x) = \frac{1}{1 + e^{-x}} \tag{1}$$

Dado que tenemos que adaptar la curva de esta función al tiempo de trayecto en condiciones de flujo libre (sin vehículos por delante), agregamos un parámetro b a la Ecuación (1), junto con un segundo término de la función inicial para hacer esto posible. Concretamente, el parámetro t_{ff} permite definir dicho tiempo de trayecto de flujo libre:

$$f(x) = \frac{1}{1 + e^{b-x}} - \frac{1}{1 + e^{b}} + t_{ff} \tag{2}$$

Finalmente, para poder adaptar la Ecuación (2) de cara a cumplir con el valor máximo real de los tiempos de viaje medidos, extendemos esta Ecuación agregando el parámetro a, y determinamos su desplazamiento correspondiente en el eje de la abscisa con el parámetro c, como se muestra en la Ecuación (3).

$$f(x) = \frac{a}{1 + e^{b-\frac{x}{c}}} - \frac{a}{1 + e^{b}} + t_{ff} \tag{3}$$

Esta ecuación es la adoptada para el análisis de regresión que sigue.

VI. Análisis del comportamiento de la congestión del tráfico

En la sección anterior, presentamos una función que se adapta mejor al comportamiento de los segmentos de nuestra ciudad objetivo a través de una regresión logística. A continuación estudiamos el comportamiento de la ciudad bajo diferentes grados de congestión vehicular. Esto se logra regulando el número de vehículos simulados siguiendo el mismo método descrito anteriormente, tomando como referencia el número estándar de vehículos durante las horas punta, e inyectando un número extra de vehículos en cada segmento de calle usando la Ecuación $\tau_{s,n} = \frac{\sigma_s}{\omega_s} \cdot \varphi_n + \lambda$ [9]. Para nuestros experimentos, variamos el número total de vehículos adicionales inyectados en la simulación de 2.271 a 34.065. Hay que tener en cuenta que el número total de segmentos para este escenario es 9.859.

Una vez obtenidos los resultados de la simulación, se realizó el proceso descrito por el Algoritmo 1. El proceso de caracterización y clasificación del comportamiento de los diferentes segmentos en la ciudad se determinó por la relación entre el número de vehículos existentes en el segmento (eje de abscisas) y el tiempo de viaje promedio en ese segmento (eje ordenado). Al concluir con la regresión logística basada en la Ecuación (3), para todos los segmentos, podemos observar patrones similares a los del enfoque cuadrático, con la mayoría de los segmentos mostrando un comportamiento creciente, y pocos segmentos que muestran una característica de flujo libre constante o única.

A. Validación de la regresión logística

Para lograr la caracterización y clasificación de los segmentos, realizamos la regresión logística utilizando la Ecuación (3), la cual ofrece un mejor ajuste que al usar la función polinómica de segundo orden, reduciendo sustancialmente el error estándar medio de 25,6477 a 7,3587, como se muestra en la Figura 2. Verificamos que la mayoría de los valores de error estándar para la regresión logística permanecen por debajo de 25 s, y los segmentos con un error estándar superior a 50 s representan menos del 1 % de los segmentos de la ciudad. Por otro lado, el error estándar asociado a la regresión cuadrática puede ir más allá de 300 s, y el error asociado a la mayoría de los segmentos de calle es mayor que 50 s.

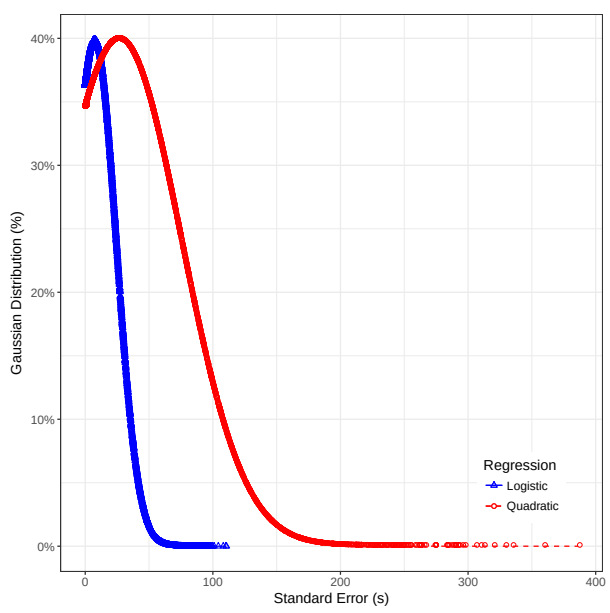


Fig. 2: Error estándar de las regresiones.

La Figura 3 muestra algunos ejemplos que ilustran cómo la regresión logística puede mejorar significativamente el error de ajuste de la curva en comparación con el enfoque estándar, basado en la regresión cuadrática. Por lo tanto, queda claro que, al utilizar una regresión logística para caracterizar el comportamiento de los segmentos, las predicciones de tiempo de viaje para diferentes volúmenes de tráfico mejoran significativamente.

B. Resultados de Clustering con regresión logística

En la sección anterior, destacamos los beneficios de adoptar una función perteneciente a la familia logística para representar adecuadamente el comportamiento de las diferentes calles de una ciudad en términos de su característica de tiempo de trayecto para diferentes volúmenes de tráfico. Una vez que se completó este paso, procedimos a realizar una clasificación apropiada de las diferentes calles de acuerdo con su caracterización. Por lo tanto, aplicamos una técnica de agrupamiento para clasificar correctamente los diferentes grupos de calles según su comportamiento.

El número de segmentos de calle para este escenario es 9.859, como se indica en la Sección VI. La representación de cada grupo caracterizado se realizó utilizando una técnica de aprendizaje automático llamada K-means [26], y utilizamos los parámetros de la regresión logística en la Ecuación (3) para realizar una categorización automática de los diferentes segmentos de la calle, asignando cada segmento a una categoría específica de acuerdo con su comportamiento. Con respecto a los parámetros utilizados como entrada para la caracterización de los segmentos, tenemos el parámetro a, que nos permite discriminar entre tendencias crecientes y constantes, ya que representa la amplitud de la curva de regresión en el eje de ordenadas. Del mismo modo, el parámetro c nos da la característica de la extensión máxima de la curva sigmoide en el eje de abscisas, que representa el número de vehículos en el segmento. Otro parámetro de entrada para el procedimiento de clasificación es t_{ff}, que representa el tiempo de viaje en un segmento cuando no hay vehículos delante, y que nos ayuda a distinguir los segmentos de acuerdo con sus velocidades de flujo libre. Finalmente, usamos un parámetro llamado $f(x)_{max}$ que representa el tiempo de viaje más alto asociado con un segmento en particular.

El método de agrupamiento de K-means identifica claramente tres agrupaciones. A continuación utili-

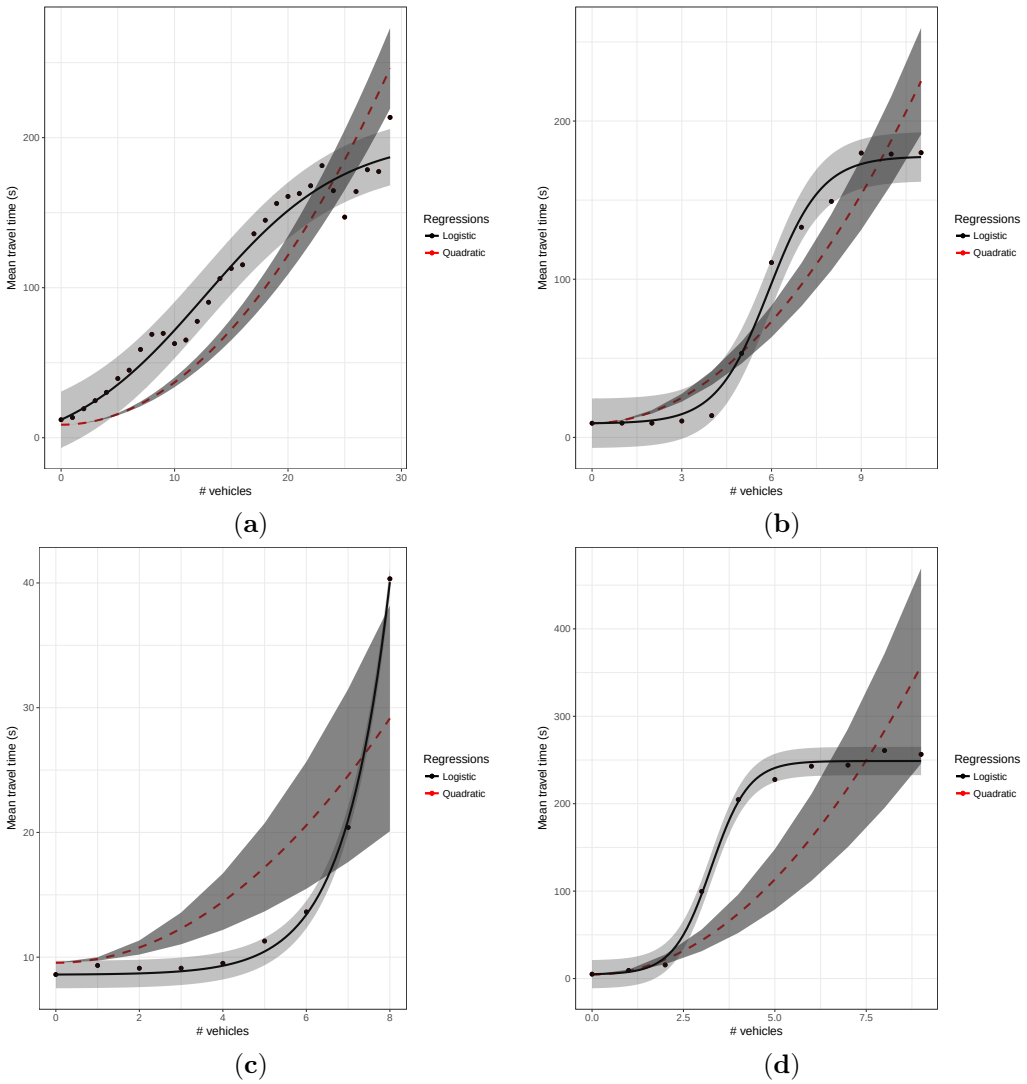


Fig. 3: Varios ejemplos donde la regresión logística supera a la regresión cuadrática.

.

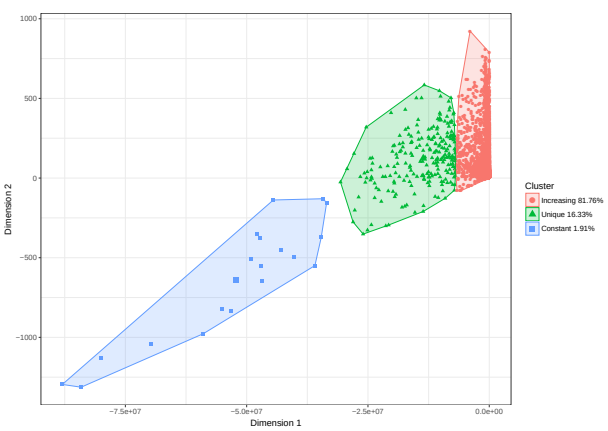


Fig. 4: Clasificación de segmentos con regresión logística por clustering.

zamos el procedimiento conocido como *análisis de componentes principales* (PCA) [27] para reducir la representación gráfica de los cuatro parámetros de entrada a dos dimensiones, y así poder entender la clasificación de los segmentos de manera visual. La Figura 4 muestra el resultado de la aplicación de K-means clustering a la ciudad de Valencia. Observamos que el porcentaje de segmentos con el comportamiento de aumento esperado es 81,76 %. Del mismo modo, podemos observar que el porcentaje de segmentos dentro de la categoría constante es 1,91 %, mientras que el 16,33 % de los segmentos de la ciudad pertenecen a la familia de segmentos únicos, que en general son segmentos bastante pequeños que se mantienen a pesar de aplicar nuestro algoritmo de reunificación de segmentos.

En general, la presencia de segmentos caracterizados por un valor constante, a pesar de que se inyectan muchos vehículos en los segmentos de la ciudad, puede ser un problema en el sentido de que tal comportamiento no es realista. Por otro lado, los segmentos únicos no reflejan los efectos de la saturación del tráfico, porque muchos de ellos todavía representan particiones muy pequeñas, como en el caso de una rotonda que no tiene un tamaño necesario, impidiendo que varios segmentos muy pequeños sean caracterizados. Por este motivo, se ha realizado un análisis más detallado de las longitudes reales de los segmentos. Descubrimos que había segmentos que tenían

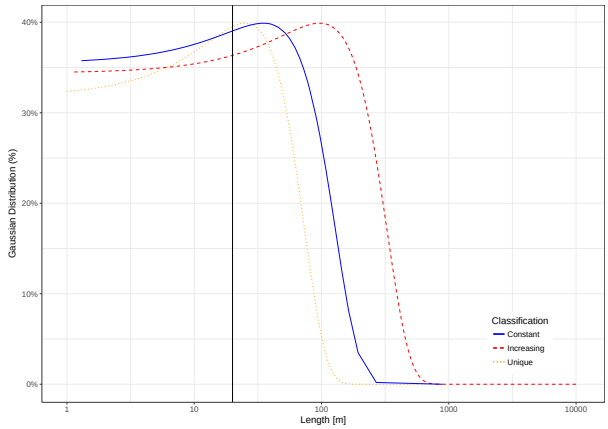


Fig. 5: Función distribución de la longitud de los segmentos.

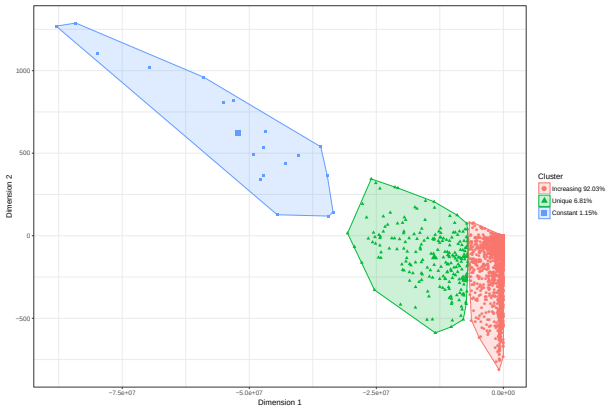


Fig. 6: Clasificación de segmentos a través de agrupamiento para la regresión logística, después de aplicar el umbral de filtrado.

una longitud menor que la longitud de un vehículo estándar. Por lo tanto, para nuestro estudio y el propósito general de predecir retrasos en el tráfico, hay que descartar dichos segmentos. De hecho, cada segmento cuyo tamaño es inferior a, al menos, la longitud del vehículo más el espacio entre vehículos, se puede descartar. Para lograr esto, optamos por el criterio propuesto por Cal y Mayor y Cárdenas [28], que es una teoría del flujo vehicular que explica el flujo, la velocidad, la densidad, el intervalo y la velocidad del vehículo, asíŋ como el espaciamiento entre vehículos. Según los autores, la ecuación fundamental del vehículo es capaz de relacionar una velocidad aproximada constante, el intervalo de tiempo libre promedio entre dos vehículos y su espaciado promedio. Por lo tanto, aplicandolo a nuestro escenario, y el resultado obtenido es de casi 20 m de longitud para cualquier segmento, lo cual nos da un umbral para filtrar cualquier segmento que mida menos de 20 m. La Figura 5 muestra que las longitudes de los segmentos de las diferentes categorías siguen una distribución gaussiana, y podemos percibir visualmente el efecto de dicho umbral de filtrado. De hecho, un alto porcentaje de segmentos que pertenecen a la familia única están por debajo de este umbral. Con respecto a las otras dos categorías, solo un pequeño porcentaje de estos segmentos cae por debajo del umbral. Por lo tanto, lo consideramos adecuado para nuestros propósitos.

Después de filtrar estos pequeños segmentos de calles, procederemos a identificar cuál es ahora el porcentaje real de segmentos que pertenecen a cada categoría aplicando nuevamente el algoritmo de agrupamiento, y recuperando su representación visual correspondiente a través del procedimiento de PCA. La Figura 6 muestra que el porcentaje de segmentos en la primera categoría crece ahora hasta el 92,03 %, y que el porcentaje de segmentos en la categoría única se reduce a tan solo 6,81 %, lo cual es significativamente pequeño. Finalmente, un número similar de segmentos de calle (1,15 %) permanece en la categoría de comportamiento constante.

La Figura 7 proporciona una visión general de la distribución geográfica real de los segmentos de calles que pertenecen a cada una de estas categorías en la ciudad de Valencia. Como se esperaba, todas las arterias principales de la ciudad pertenecen a la categoría “creciente”, ya que solo los segmentos muy pequeños y remotos, ubicados en calles secundarias, pertenecen a las otras dos categorías.

C. Comportamiento de congestión del tráfico en un escenario Hotspot

En esta sección, nos centramos en situaciones en las que un evento público aglutina a un gran número de personas en un área restringida, lo que hace que la ciudad experimente un efecto de congestión heterogéneo. Para este escenario, que denominamos “hotspot”, hemos elegido un área de 270 m de radio centrada en el estadio de fútbol de Mestalla, y que tiene 106 rutas predichas diferentes que pasan cerca. La estrategia para lograr nuestro objetivo es inyectar gradualmente vehículos en este escenario en un rango de 100 a 10.000, y que están dispersos en un área que incluye un total de 887 segmentos. Por lo tanto, este escenario combina "vehículos regulares", que pasan por esa área como en un día cualquiera (saliendo y llegando), y “vehículos con puntos de conexión”, que salen del área congestionada y se trasladan a cualquier destino aleatorio. Nuestro objetivo es estudiar los efectos de dicha congestión asimétrica en comparación con la situación estudiada anteriormente, donde la congestión es más homogénea. Concretamente, queremos averiguar si dicha congestión de tráfico localizada puede generar condiciones que permitan realizar una mejor caracterización de los diferentes segmentos de calle en términos de su curva de predicción para los tiempos de trayecto.

Para este escenario, aplicamos el Algoritmo 1 a los segmentos que pertenecen a la zona congestionada, y obtuvimos la predicción de los tiempos de viaje. Aplicado la regresión logística (ver Ecuación (3)), y obtuvimos el comportamiento de los diferentes segmentos de calle. De nuevo encontramos que los comportamientos de los segmentos dentro de esta área pertenecen a las mismas tres categorías defini-

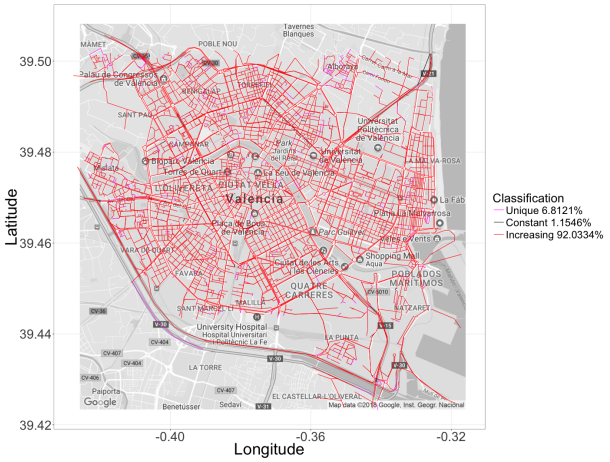


Fig. 7: Distribución geográfica de la clasificación de segmentos.

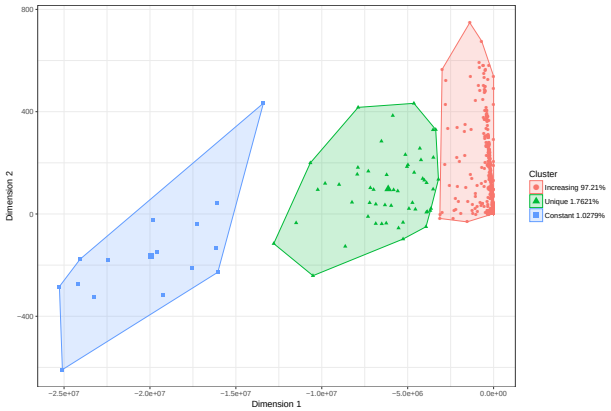


Fig. 8: Clasificación de segmentos mediante clustering para la regresión logística en el escenario hotspot.

das anteriormente: tendencia creciente, valor único, y valor constante. Como se explicó antes, la presencia de microsegmentos en estos escenarios persiste. Por lo tanto, nuevamente aplicamos un filtro a la longitud de estos segmentos para descartar aquellos que son excesivamente pequeños, ya que son irrelevantes para nuestro estudio. A continuación procedimos a realizar la clasificación automática a través del algoritmo de Clustering junto con el PCA, para visualizar esos grupos en un espacio bidimensional. Como se muestra en la Figura 8, la técnica de Clustering muestra ahora que el 97,21 % de los segmentos de calle pertenecen a la categoría principal, lo que significa que la mayoría de los segmentos de calle ya se pueden caracterizar correctamente en términos de comportamiento de tiempo de viaje para diferentes niveles de congestión vehicular. Los dos grupos restantes están asociados a un porcentaje muy bajo de segmentos: 1,7621 % para la categoría de valor único, y 1,0279 % para el caso de tiempo constante. Del mismo modo, podemos observar en la Figura 9 la ubicación geográfica de los segmentos dentro del escenario estudiado, diferenciados según su comportamiento. Podemos ver que los segmentos para los cuales tenemos una caracterización de retardo deficiente (caso de valores únicos o constantes) en realidad no son del todo relevantes desde una perspectiva global, ya que para la mayoría se puede obtener una visión clara del comportamiento del viaje, y dicha caracterización por segmento se usa como entrada para un sistema de planificación de rutas global.

VII. Conclusiones y trabajo futuro

Tener un modelo de tráfico realista para una ciudad específica es un requisito clave para obtener resultados de simulación significativos cuando problemas como la densidad del tráfico y los patrones de tráfico pueden tener un impacto en las conclusiones derivadas de los experimentos. Lograr dichos modelos realistas pasa generalmente por describir el tráfico en términos de matrices Origen-Destino (O-D). Además, si está orientado al desarrollo de soluciones avanzadas de gestión de tráfico, es necesario tener una comprensión más profunda de cómo se distribuye el tráfico en una ciudad en particular, lo que básicamente requiere realizar un análisis y una clasificación correctos de dicho tráfico.

El punto de partida de este documento fue un modelo de tráfico realista para Valencia derivado de un trabajo anterior. A partir del mismo, la contribución de este trabajo ha sido la caracterización de todos los segmentos de calle de Valencia en términos de tiempos de trayecto para diferentes grados de congestión. Para lograr esta caracterización, comenzamos por procesar el mapa del área objetivo para fusionar segmentos de la misma calle siempre que se detectara una fragmentación innecesaria y pudiera revertirse. A continuación, realizamos experimentos de simulación utilizando SUMO para recuperar los tiempos de trayecto de los vehículos ante diferentes grados de saturación de tráfico para cada segmento. Finalmente, utilizando diferentes estrategias de regresión, realizamos un ajuste para obtener una expresión que nos permitió caracterizar estos tiempos de viaje.

Una vez que todos los segmentos se caracterizaron, nuestra siguiente contribución fue agrupar de manera automática los diferentes segmentos en clústeres según su comportamiento en cuanto a tiempo de trayecto. Para ello aplicamos la técnica de K-means para generar los clústeres, seguido de un análisis de componentes principales para extraer las principales características de agrupamiento que permiten una representación visual. Los resultados del proceso de agrupación definen claramente tres categorías: segmentos con retrasos de tráfico incrementales (la mayoría), segmentos con retrasos constantes (las cargas típicas no causan congestión), y resultados de valor único correspondientes a pequeños segmentos raramente visitados por vehículos. Complementamos este estudio con un análisis de las longitudes de segmento para filtrar aquellos segmentos que son demasiado pequeños y, por lo tanto, no son representativos para nuestro análisis de tráfico. También mostramos cómo la caracterización del segmento de calle podría mejorarse al causar niveles de congestión muy altos en una determinada área, situación en la que más del

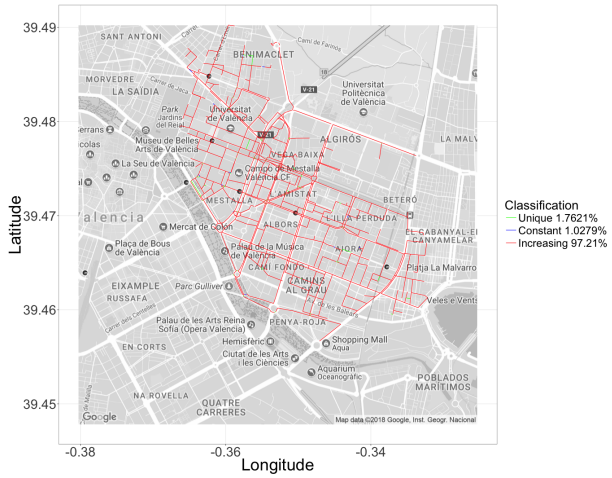


Fig. 9: Distribución geográfica de los segmentos en los diferentes grupos para el escenario hotspot.

97 % de los segmentos podrían caracterizarse adecuadamente.

Como trabajo futuro, planeamos desarrollar una plataforma de gestión de tráfico centralizada que, en función de la caracterización del retardo de viaje por segmento que se proporciona en este documento, pueda minimizar globalmente los tiempos de viaje del vehículo al contabilizar la congestión y realizar un equilibrado de carga. Una vez que se haya desarrollado la plataforma, se estudiará el impacto de tener diferentes tipos de vehículos, con sus características respectivas, como la velocidad máxima, los horarios de tráfico permitidos, y carriles especializados para vehículos de servicio público. Esto nos permitirá conocer la eficiencia de la plataforma al considerar los distintos tipos de vehículos disponibles (por ejemplo, autobuses y camiones) en el flujo de tráfico general.

Agradecimientos

Este trabajo ha sido financiado por el Ministerio de Ciencia, Innovación y Universidades, Programa Estatal de Investigación, Desarrollo e Innovación Orientada a los Retos de la Sociedad, Proyectos I+D+I 2018, España (RTI2018-096384-B-I00), y por el Programa de Becas SENESCYT de la República del Ecuador.

Referencias

[1] Jabali, O.; Woensel, T.; de Kok, A.G. Analysis of travel times and CO_2 emissions in time-dependent vehicle routing. *Prod. Oper. Manag.* **2012**, *21*, 1060–1074, doi:10.1111/j.1937-5956.2012.01338.x.

[2] Vallati, M.; Magazzeni, D.; De Schutter, B.; Chrpa, L.; McCluskey, T.L. Efficient Macroscopic Urban Traffic Models for Reducing Congestion: A PDDL + Planning Approach. In Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence (AAAI-16) Phoenix, AZ, USA, 12–17 February 2016; pp. 3188–3194.

[3] Pozanco, A.; Fernández, S.; Borrajo, D. Urban Traffic Control Assisted by AI Planning and Relational Learning. In *ATT@ IJCAI*, 2016.

[4] Xie, X.F.; Smith, S.F.; Barlow, G.J. Schedule-Driven Coordination for Real-Time Traffic Network Control. In Proceedings of the Twenty-Second International Conference on International Conference on Automated Planning and Scheduling (ICAPS 2012), Palaiseau, France, 23–27 July 2012; ACM: New York, NY, USA, 2013; pp. 323–331.

[5] Djahel, S.; Doolan, R.; Muntean, G.M.; Murphy, J. A communications-oriented perspective on traffic management systems for smart cities: Challenges and innovative approaches. *IEEE Commun. Surv. Tutor.* **2015**, *17*, 125–151, doi:10.1109/COMST.2014.2339817.

[6] Zambrano, J.L.; Calafate, C.T.; Soler, D.; Cano, J.C.; Manzoni, P. Using real traffic data for its simulation: Procedure and validation.In Proceedings of the 2016 International IEEE Conferences on Ubiquitous Intelligence & Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People, and Smart World Congress (UIC/ATC/ScalCom/CBDCom/IoP/SmartWorld), Toulouse,France, 18–21 July 2016; pp. 161–170, doi:10.1109/UIC-ATC-ScalCom-CBDCom-IoP-SmartWorld.2016.0045.

[7] Calafate, C.T.; Soler, D.; Cano, J.C.; Manzoni, P. Traffic management as a service: The traffic flow pattern classification problem. *Math. Probl. Eng.* **2015**, *2015*, doi:10.1155/2015/716598.

[8] Nguyen, T.V.; Krajzewicz, D.; Fullerton, M.; Nicolay, E. DFROUTER-Estimation of Vehicle Routes from Cross-Section Measurements. In *Modeling Mobility with Open Data*; Springer: Berlin/Heidelberg, Germany, 2015; pp.3–23, ISBN 978-3-319-15024-6.

[9] Zambrano-Martinez, J.L.; Calafate, C.T.; Soler, D.; Cano, J.C. Towards realistic urban traffic experiments using DFROUTER: Heuristic, validation and extensions. *Sensors* **2017**, *17*, 2921, doi:10.3390/s17122921.

[10] Zambrano-Martinez, J.L.; Calafate, C.T.; Soler, D.; Cano, J.C.; Manzoni, P. Analysis and Classification of the Vehicular Traffic Distribution in anUrban Area. In *Adhoc, Mobile, andWireless Networks*; Puliafito, A., Bruneo, D., Distefano, S., Longo F., Eds.; Springer: Cham, Switzerland, 2017; pp. 121–134, doi:10.1007/978-3-319-67910-5_10.

[11] Behrisch, M.; Bieker, L.; Erdmann, J.; Krajzewicz, D. SUMO—Simulation of urban mobility: Anoverview. In Proceedings of the Third International Conference on Advances in System Simulation. ThinkMind (SIMUL 2011), Barcelona, Spain, 23–28 October 2011; IARIA XPS Press: Kobenhavn, Denmark, 2011.

[12] Lieu, H. *Revised Monograph on Traffic Flow Theory*; US Department of Transportation Federal Highway Administration: Washington, DC, USA, 2003.

[13] Zhang, X.; Rice, J.A. Short-term travel time prediction. *Transp. Res. C Emerg. Technol.* **2003**, *11*, 187–210, doi:10.1016/S0968-090X(03)00026-3.

[14] Guo, J.; Huang, W.; Williams, B.M. Adaptive Kalman filter approach for stochastic short-term traffic flow rate prediction and uncertainty quantification. *Transp. Res. C Emerg. Technol.* **2014**, *43*, 50–64, doi:10.1016/j.trc.2014.02.006.

[15] Van Hinsbergen, C.P.; Schreiter, T.; Zuurbier, F.S.; Van Lint, J.W.C.; Van Zuylen, H.J. Localized extended kalman filter for scalable real-time traffic state estimation. *IEEE Trans. Intell. Transp. Syst.* **2012**, *13*, 385–394, doi:10.1109/TITS.2011.2175728.

[16] Zhang, X.; Onieva, E.; Perallos, A.; Osaba, E.; Lee, V. Hierarchical fuzzy rule-based system optimized with genetic algorithms for short term traffic congestion prediction. *Transp. Res. C Emerg. Technol.* **2014**, *43*, 127–142, doi:10.1016/j.trc.2014.02.013.

[17] Onieva, E.; Milanés, V.; Villagra, J.; Pérez, J.; Godoy, J. Genetic optimization of a vehicle fuzzy decision system for intersections. *Expert Syst. Appl.* **2012**, *39*, 13148–13157, doi: 10.1016/j.eswa.2012.05.087.

[18] Hodge, V.J.; Krishnan, R.; Jackson, T.; Austin, J.; Polak, J. Short-Term Traffic Prediction Using a Binary Neural Network. In Proceedings of the 43rd Annual UTSG Conference, York, UK, 5–7 January 2011.

[19] Habtie, A.B.; Abraham, A.; Midekso, D. Artificial Neural Network Based Real-Time Urban Road Traffic State Estimation Framework. In *Computational Intelligence in Wireless Sensor Networks*; Springer: Cham, Switzerland, 2017; pp. 73–97, doi:10.1007/978-3-319-47715-2_4.

[20] Porikli, F.; Li, X. Traffic congestion estimation using HMM models without vehicle tracking. InProceedings of the 2004 IEEE Intelligent Vehicles Symposium, Parma, Italy, 14–17 June 2004; pp. 188–193, doi:10.1109/IVS.2004.1336379.

[21] Kunt, M.M.; Aghayan, I.; Noii, N. Prediction for traffic accident severity: Comparing the artificial neural net-

work, genetic algorithm, combined genetic algorithm and pattern search methods. *Transport* **2011**, *26*, 353–366, doi:10.3846/16484142.2011.635465.

[22] Kerner, B.S.; Rehborn, H.; Aleksic, M.; Haug, A. Traffic prediction systems in vehicles. In Proceedings of the 2005 IEEE Intelligent Transportation Systems, Vienna, Austria, 16 September 2005; pp. 72–77, doi:10.1109/ITSC.2005.1520056.

[23] Basnayake, C. Automated traffic incident detection with GPS equipped probe vehicles. In Proceedings of the the 17th International Technical Meeting of the Satellite Division of the Institute of Navigation, LongBeach,CA, USA, 21–24 September 2004; pp.1–10.

[24] Varga, A.; Hornig, R. An overview of the OMNeT++ simulation environment. In Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops, Marseille, France, 3–7 March 2008.

[25] Menard, S. *Applied Logistic Regression Analysis*; SAGE Publications: Thousand Oaks, CA, USA , 2018; Volume106, ISBN 9780761922087.

[26] Jain, A.K. Data clustering: 50 years beyond K-means. *Pattern Recognit. Lett.* **2010**, *31*, 651–666, doi:10.1016/j.patrec.2009.09.011.

[27] Li, J.; Linear, R.R. Principal Component Analysis. In *Multivariate Statistics*; Springer: Berlin, Germany, 2014; Volume 487, pp. 163–183, doi:10.1007/978-0-387-73508-5_9.

[28] Cal y Mayor Reyes Spíndola, R.; Cárdenas Grisales, J. *Ingeniería de Tránsito: Fundamentos y Aplicaciones*; Alfaomega Grupo Editor: Mexico D.F., Mexico, 2010.(In Spanish).

Medición de la eficiencia industrial mediante dispositivos de bajo coste

Angel C. Herrero[1], Francisco J. Martinez[1], Piedad Garrido[1], Julio A. Sangüesa[2]

Resumen— **La cuarta revolución industrial, conocida como Industria 4.0, es una realidad creada a partir de la evolución e innovación tecnológica. Aspectos como la conectividad entre máquinas, personas y productos, el análisis en tiempo real de multitud de variables y el uso de técnicas basadas en Inteligencia Artificial, entre otras, han dotado a la industria de una enorme versatilidad, permitiendo la mejora continua en los procesos. Por contra, la implantación de novedades tecnológicas en la industria suele acarrear una gran inversión y quizá sea éste el motivo principal por el que en muchos ámbitos, no se haya llegado a implantar. En el presente trabajo, proponemos un dispositivo de bajo coste basado en el uso de Rasperry Pi, capaz de calcular el índice de la efectividad global de la maquinaria industrial (OEE). El uso de nuestro sistema permitirá mejorar la capacidad de reacción de la empresa, pues posibilitará detectar los aspectos que se deben mejorar para aumentar globalmente la productividad. Los resultados obtenidos demuestran que el sistema propuesto es robusto y que las mediciones son precisas, habiendo reducido notablemente el coste necesario, especialmente si lo comparamos con otras soluciones similares basadas en dispositivos propietarios.**

Palabras clave— **Industria 4.0, OEE, dispositivos de bajo coste, Internet de las Cosas, Raspberry Pi.**

I. Introducción

EL desarrollo actual de la industria apunta hacia una estructura más flexible y modular, capaz de adaptarse a las necesidades cambiantes del mercado. Para satisfacer estas necesidades, han aparecido algunos nuevos paradigmas relacionados con el entorno industrial como la Fabricación en la Nube (*Cloud Manufacturing* (*CM*)), la Industria 4.0, la Fabricación Orientada a Servicios y Sistemas de Producción Ciber Físicos (*Cyber-Physical Production Systems* (*CPPS*)). Además, debido al desarrollo de la Internet Industrial de las Cosas (IIoT), todas estas alternativas han sufrido un crecimiento exponencial en los últimos años [1].

El concepto de Industria 4.0 es relativamente reciente y se refiere a la cuarta revolución industrial que consiste en la introducción de las tecnologías digitales en la industria. Estas permiten que dispositivos y sistemas colaboren entre ellos y con otros, permitiendo modificar los productos, los procesos y los modelos de negocio [2]. La Figura 1 muestra los elementos fundamentales que intervienen en el concepto de Industria 4.0.

Gracias a la conexión entre operaciones físicas y los sistemas informáticos, los procesos de fabricación se transforman, personalizan y se orientan al consumidor final. Esto permite realizar fabricaciones en serie con un menor número de unidades por producto, disminuir los tiempos y el número de operaciones a realizar, facilitando cambios de producción de un producto concreto a otro de diferentes características en tiempos mínimos, dando paso a una fabricación con altos niveles de customización. Para lograr un control total del proceso, los productos son marcados con códigos que incluyen su trazabilidad, siendo uno de los objetivos de esta codificación, la detección de fallos en el modelo productivo. Con ello se eliminan los defectos allí donde se generan, garantizando la calidad, conocimiento de todos los pasos realizados para la consecución final del producto y la información necesaria para intervenir en tiempo real si fuese preciso, así como proveer al consumidor de toda la información del producto terminado.

Los elementos de control, monitorización, sensores, así como la amplia tipología de dispositivos y equipos utilizados en la Industria Conectada se caracterizan, en su mayor parte, por ser sistemas exclusivos de los fabricantes, por el encarecimiento que conlleva su implantación, así como por la baja estandarización de protocolos de comunicación e interoperabilidad. Debido a estos inconvenientes, el presente trabajo presta especial atención a la utilización de sistemas con funcionalidades similares a otros propietarios de reconocido prestigio, pero con la característica principal de ser de menor coste. Este paradigma, todavía incipiente en el ámbito industrial, pretende dotar a los procesos industriales de dispositivos, programación y *hardware* con las propiedades y funcionalidades demandadas por y para la digitalización de la industria, pero a un coste mucho menor.

En la tecnología de bajo coste se incluyen microcontroladores, sensores y otros dispositivos *Hardware*, tales como Raspberry Pi o Arduino, que permiten por sí solos o combinados con otros periféricos complementarios, la monitorización y el control de variables, análisis de datos, actuar como servidores y en definitiva, conectar el estado de las máquinas para su análisis en tiempo real o diferido.

Mediante la introducción del Internet de las Cosas (IoT), en las áreas en las que se utilizaban las TIC de forma tradicional, se ha visto mejorar su alcance y consolidar su uso al abaratar objetos, conectividad y almacenaje de datos de forma abierta. Los sistemas de control y telemedida tradicionalmente desarrollados en entornos cerrados y de alto precio, ahora pueden ser sustituidos por soluciones de más bajo coste, con arquitecturas, protocolos de comunicación, y sistemas de representación de datos abiertos [3].

[1]Dpto. de Informática e Ingeniería de Computadores, Universidad de Zaragoza, e-mail: acherrero, f.martinez, piedad@unizar.es

[2]Centro Universitario de la Defensa, Zaragoza, e-mail: jsanguesa@unizar.es

Fig. 1. Pilares estratégicos en la Industria 4.0 [4]

En este artículo se propone un sistema, basado en dispositivos de bajo coste, utilizando como elemento principal una *Raspberry Pi*, especialmente diseñado para la monitorización de procesos industriales y la consecución de una mejora en los parámetros de producción. El objetivo principal de esta propuesta es monitorizar determinadas variables en tiempo real y mediante un dispositivo de bajo coste, que contribuyan al cálculo de la efectividad global de la maquinaria industrial (*Overall Equipment Effectiveness*, (*OEE*)). Para validar el correcto funcionamiento de nuestra propuesta, lo hemos puesto en marcha en un entorno real, más en concreto en una planta de loncheado de materia prima.

El artículo está organizado de la siguiente manera: en la Sección II se incluyen algunos artículos relacionados con nuestro trabajo. En concreto, varias propuestas basadas en dispositivos de bajo coste. En la Sección III se presenta en detalle la propuesta planteada para la medición de rendimiento industrial a través del cálculo en tiempo real del OEE, así como la Arquitectura del sistema, las conexiones del dispositivo y el flujo de señales. La Sección IV muestra el formato de representación de los datos, así como los resultados obtenidos en las mediciones y la validación de nuestro sistema. Por último, en la Sección V se exponen las conclusiones más relevantes.

II. Trabajos Relacionados

El alto coste de los sistemas de medición, monitorización y control impide, en cierta manera, el despliegue efectivo de la conocida como Industria 4.0. De hecho, su implementación en los procesos industriales, resulta difícil de justificar financieramente en numerosas ocasiones. Los aspectos que necesitan de la estandarización para la integración de la Industria 4.0 son principalmente los protocolos de comunicaciones, los conectores e interfaces físicas, la interoperabilidad y ciberseguridad, así como las plataformas de gestión y tecnologías I4.0 [5]. De este modo, el desarrollo de nuevos estándares pretende que la industria pueda tener mejores opciones para automatizar un proceso y así crear sistemas más flexibles y económicos a la hora de ponerlos en marcha, ya que la solución final no estará limitada por el proveedor.

La utilización de dispositivos de bajo coste pretende facilitar la instalación y desarrollo de microcontroladores, minicomputadoras, así como el tratamiento de datos mediante *BigData* y la aplicación de algoritmos de *Machine Learning*, permitiendo la implantación de nuevos sistemas en la industria. A pesar de la continua aparición de dispositivos de bajo coste en el mercado que pueden ser dedicados a la industria, no hay una implantación significativa de este tipo de dispositivos orientados a la monitorización de la maquinaria industrial. Como veremos a continuación, a la hora de afrontar estos retos, diversos autores han utilizado Raspberry Pi para ofrecer soluciones a la Industria 4.0.

Una de estas soluciones, se basa en el uso de una Raspberry Pi 2 Modelo B [6]. En concreto, los autores proponen el control de sistemas de automatización haciendo uso de éste y otros dispositivos de bajo coste, así como *software* libre. En el sistema propuesto, emplean una tarjeta de expansión de entrada/salida PiFace Digital 2 [7]. Los dos dispositivos ensamblados (es decir, la Raspberry y la PiFace), constituyen un Controlador Lógico Programable (PLC) de bajo coste. El control de la planta, a nivel de software, se implementa utilizando CoDeSys [8], que proporciona tanto el sistema de desarrollo como el de ejecución. Sin embargo, llama la atención que las pruebas de control de la planta se realizaron mediante el uso de simulación. Otros autores como Caiza y García [9] han aprovechado las virtudes de la Raspberry Pi 3 Modelo B para la implantación de un sistema distribuido de bajo coste, usando la norma IEC-61499 en la estación de clasificación y manipulación del MPS-500, una estación de producción flexible-compatible, modular y versátil de la marca FESTO. El IEC-61499 estandariza un entorno de programación para sistemas distribuidos y tiene un alto nivel de versatilidad para el diseño de sistemas, pues combina software y hardware de manera independiente. Al igual que la propuesta anterior, los autores no aplicaron su solución en un entorno real.

En lo relativo a la monitorización, Toapanta [10] presenta un sistema de monitorización de variables de entorno en procesos de fabricación, basado en el paradigma de los sistemas ciberfísicos industriales. Más en detalle, para su desarrollo seleccionó componentes de bajo coste, especialmente una Raspberry Pi 2 Modelo B. Combinando diferentes tecnologías de bajo coste, May *et al.* [11] realizan el desarrollo de una red de monitorización para sistemas fotovoltaicos basada en el uso de la tecnología inalámbrica ZigBee, microprocesadores ARM de 32 bits y una Raspberry Pi. El sistema propuesto está formado por un módulo de sensorización diseñado para la medición y transmisión de los parámetros de temperatura,

Fig. 2. Principales componentes de la efectividad global del equipo (OEE)

voltaje y corriente de los paneles fotovoltaicos. Dicho módulo se comunica con la Raspberry Pi que realiza las funciones de coordinación central y servidor web. Así, el módulo ZigBee transmite los parámetros a la Raspberry, que generará una base de datos con los valores recibidos, además de asignarles fecha y hora. Todos los datos actualizados pueden ser visualizados desde una aplicación web desarrollada, mediante cualquier ordenador o dispositivo móvil. Por otra parte, Párraga y Vega [12] proponen un sistema electrónico capaz de administrar el consumo energético a nivel residencial. Los dispositivos utilizados son una tarjeta Arduino UNO y una Raspberry Pi 3, además de diversos sensores y actuadores. Es preciso remarcar que la propuesta está orientada al ámbito doméstico.

Como hemos presentado, existen diferentes trabajos que usan dispositivos de bajo coste, fundamentalmente para la monitorización, el uso de sistemas lógicos programables y la regulación. No obstante, gran parte de las propuestas han sido validadas mediante simuladores, o han sido aplicados en entornos domésticos. En nuestro trabajo, sin embargo, hemos implementado nuestro sistema en un entorno real. En concreto, el trabajo que presentamos se ha implantado en 6 líneas de loncheado de una industria agroalimentaria. Gracias a nuestro sistema, podemos monitorizar y obtener en tiempo real y de forma remota, los datos relativos al rendimiento de la maquinaria.

III. Propuesta para la medición del OEE

En la actualidad, uno de los sectores más interesantes para la implantación de sistemas y tecnologías propias de la Industria 4.0 es el sector agroalimentario. Gracias a la llamada *Food Industry* 4.0, somos capaces de producir un mayor volumen de alimentos, así como de adaptar su producción más y mejor a los nichos de consumo y a las necesidades concretas del consumidor. Las nuevas tecnologías posibilitan centrar en el consumidor todas las fases del proceso de diseño, fabricación y distribución de producto, aumentando por tanto, las posibilidades de éxito [13].

El objetivo principal de nuestra propuesta es la medición de los parámetros que contribuyen al cálculo del OEE, en tiempo real y mediante dispositivos de bajo coste. La herramienta de medición OEE se desarrolló a partir del concepto Total Productive Maintenance (TPM) propuesto por Nakajima [14]. El objetivo del TPM era lograr cero fallos y defectos provocados por los equipos, generando una mejora directa en la tasa de producción, la reducción de los costes e inventario, así como el aumento de la productividad laboral. Relacionado con lo anterior, el OEE se define como una medida del rendimiento total del equipo, es decir, el grado en que el equipo está haciendo lo que se supone que debe hacer [15]. Clasifica las principales pérdidas o las razones de un rendimiento deficiente y, por lo tanto, proporciona la base para establecer prioridades de mejora y comienzo del análisis del origen de los defectos.

El OEE es una métrica de gran valor para el análisis de la producción, ya que permite la evaluación y medida de la productividad en los procesos. La Figura 2 muestra los principales componentes que se tienen en cuenta para el cálculo del OEE:

- Disponibilidad: para medirla, tenemos en cuenta los siguientes tiempos: (i) Tiempo de Producción Planificado (TPP), que se determina restándole a las horas totales programadas, los tiempos perdidos por paros planificados, y (ii) Tiempo Operativo (TO), que es el tiempo que resulta de descontar del TPP los tiempos perdidos por inactividad.
- Rendimiento: para su medida tenemos en cuenta: (i) el Tiempo Operativo Neto (TON), cuyo valor se corresponde con el TO, puesto que se parte de ese tiempo para realizar las operacio-

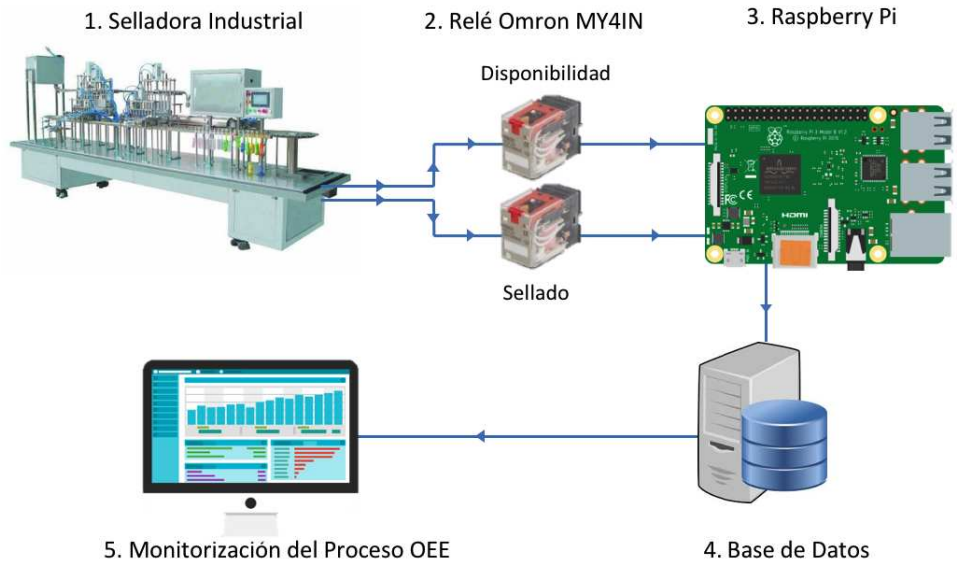


Fig. 3. Arquitectura del Sistema para cada línea de la planta

nes, y (ii) el Tiempo Operativo Real (TOR), que se obtiene descontando al TO, las pérdidas de velocidad debidas a paradas menores.

- Calidad: para estimar este componente, tendremos en cuenta: (i) el Tiempo Productivo Neto (TPN), que es el que realmente se dispone para realizar la producción y cuyo valor coincide con el TOR, y el (ii) Tiempo Productivo Real (TPR), que es el tiempo realmente efectivo, y que se obtiene al descontar el tiempo perdido debido a defectos de fabricación.

Los resultados del OEE permiten la comparación y observación de unidades de producción en diferentes industrias. El OEE proporciona datos significativos que posibilitan reducir el tiempo de trabajo y aumentar la optimización de los recursos, por lo que se utiliza para aumentar el rendimiento del sistema y mejora continua del trabajo. Además, puede revelar casos de alta demanda de trabajo y baja producción, lo que posibilitaría adoptar una mejor actuación en la industria [16].

La Ecuación 1 muestra cómo se calcula el OEE.

$$OEE = Disponibilidad \cdot Rendimiento \cdot Calidad \quad (1)$$

donde: $Disponibilidad = \frac{TO}{TPP}$, $Rendimiento = \frac{TOR}{TON}$, y $Calidad = \frac{TPR}{TPN}$.

A continuación, pasamos a comentar en detalle nuestra propuesta, haciendo especial hincapié en su arquitectura, el diagrama de flujo, así como las señales y conexiones de los diferentes componentes.

A. Arquitectura del sistema

Nuestra propuesta se ha centrado en la medida del OEE en una planta de procesado de productos cárnicos, en concreto en las líneas en las que se lonchea la materia prima, se envasa y se realiza el sellado termoplástico de los envases.

La configuración del sistema desarrollado para la medición de las variables de eficiencia productiva se muestra en la Figura 3. En este esquema se pueden diferenciar claramente los cinco elementos más importantes que intervienen en la operación de medición.

1. Selladora Industrial. Una vez que se ha loncheado y envasado la materia prima, esta máquina es la encargada de realizar las operaciones de sellado en los envases. Este elemento pertenece a la cadena de producción del proceso productivo y es el emisor de las diferentes señales que recogerá el dispositivo de bajo coste diseñado.
2. Relé Omron MY4IN. Es el dispositivo electromagnético que se estimula a partir de la corriente eléctrica recibida por la selladora industrial en sus diferentes estados. Éste abrirá o cerrará el circuito con su correspondiente envío de señal. En nuestro sistema existen dos unidades encargadas de recibir las señales de disponibilidad y sellado (para medir el rendimiento) de manera independiente, remitiendo cada una de las métricas a la Raspberry Pi.
3. Raspberry Pi. Es el dispositivo encargado de recibir las señales del sistema y procesarlas digitalmente, para su posterior envío a la base de datos. La Raspberry, además de tratar las señales realiza la función de servidor, permitiendo el acceso remoto desde cualquier dispositivo conectado a Internet, para la visualización y monitorización de las líneas de producción. La utilización de Raspberry Pi se debe, no solo a su bajo coste, sino por la necesidad de realizar inserciones de los parámetros procesados en la base de datos y por la capacidad para monitorizar estos datos a través del servidor programado en el propio dispositivo.
4. Base de Datos. La base de datos se encarga de almacenar todas las mediciones realizadas y pro-

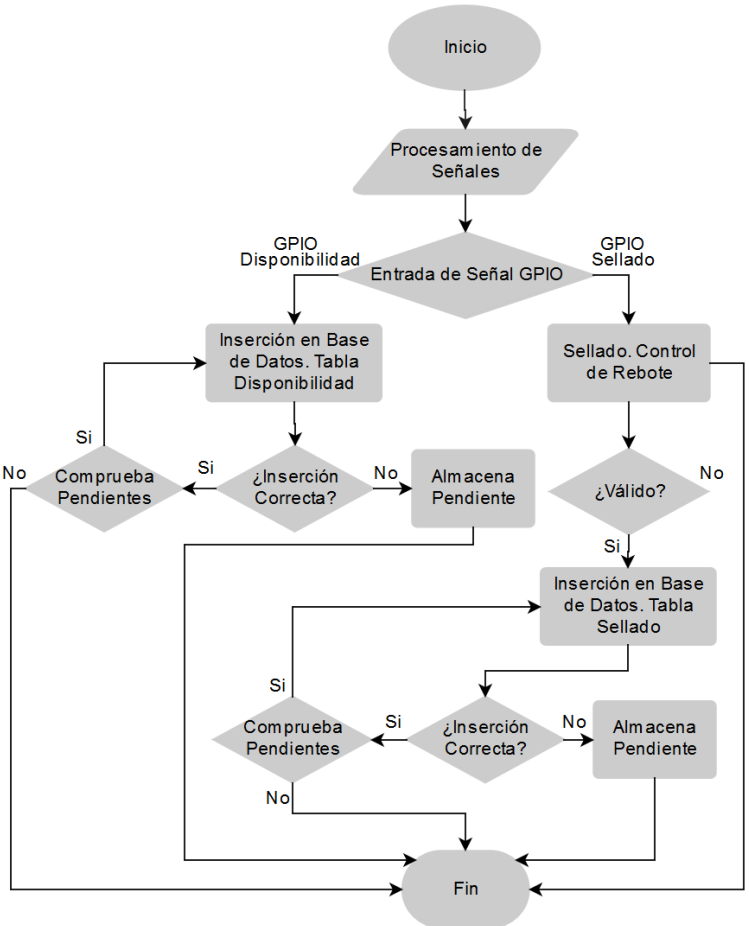


Fig. 4. Diagrama de flujo del sistema propuesto.

cesadas por la Raspberry Pi. De esta forma, podremos obtener los datos históricos de las medidas recogidas para su posterior tratamiento y análisis, en caso de ser necesario.

5. Monitorización del OEE. El último elemento de nuestro sistema estará formado por los diferentes equipos desde los que cualquier usuario con privilegios podrá visualizar en tiempo real las distintas métricas relacionadas con la efectividad global (es decir, los ratios de disponibilidad, de rendimiento y de calidad).

El sistema propuesto, por tanto, recoge y procesa las variables aportadas por la selladora industrial. En concreto, se registra la disponibilidad (es decir, cuándo la selladora está, o no, disponible para trabajar o está trabajando) y la operación de sellado (es decir, cuándo baja el pistón sellador).

B. Diagrama de flujo de nuestro sistema

La Figura 4 muestra el proceso que ocurre cuando la Raspberry Pi recibe una señal de entrada desde la selladora. El dispositivo está programado con el código necesario para poder realizar el procesado de las señales procedentes de la máquina y su posterior inserción en la base de datos. La recepción de señales en la Raspberry Pi se realiza a través de dos de sus pines *General Purpose Input/Output* (GPIO), y éstas son clasificadas en función de su procedencia (GPIO de Sellado o GPIO de Disponibilidad), de tal manera que cada señal sea tratada de forma adecuada.

Las señales procedentes de Disponibilidad se insertan directamente en una tabla de la Base de Datos creada específicamente para recoger esta información. Estas señales son de tipo High (1) o Low (0), es decir, indican si la máquina está lista o no para poder trabajar. Tras la tarea de inserción, se comprueba si la transacción ha sido correcta. En caso de no ser correcta (normalmente debido a falta de conectividad), los datos enviados se almacenarán como pendientes para su correcto procesamiento una vez que se restablezca la conexión. Si la inserción de Disponibilidad se ha realizado correctamente, se comprueba que no existen registros pendientes de inserción, y se finaliza el proceso.

Cuando las señales provienen del GPIO de Sellado, éstas se someten a un control de rebotes, que consiste en detectar si el tiempo entre dos señales contiguas ha sido suficiente para considerar que verdaderamente es una señal de sellado, o en caso contrario, las desestimamos al considerarlas no válidas (consideradas como ruido). El tiempo mínimo entre señales es de 6 segundos ya que la selladora industrial puede realizar un máximo de 10 sellados por minuto. Si la señal es válida, se realiza la inserción en la base de datos, en la tabla correspondiente de sellado. Cuando el resultado de la inserción no es correcto se almacenará

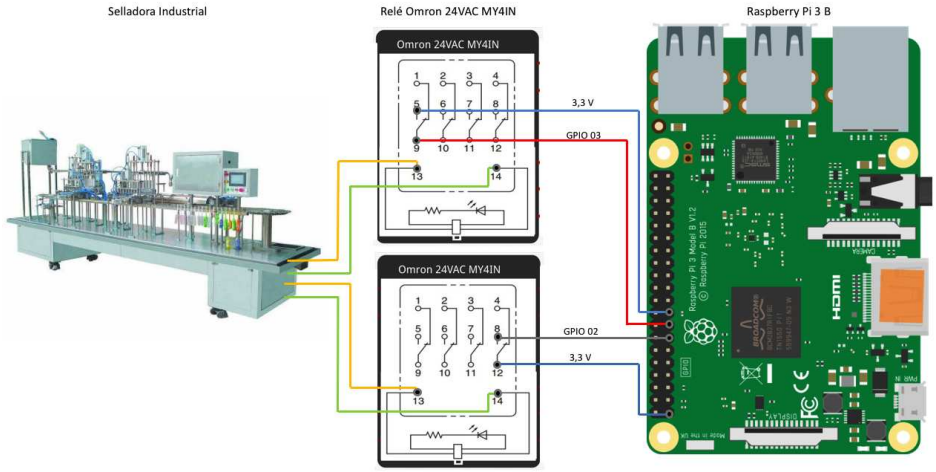


Fig. 5. Cableado de los relés con la Raspberry Pi y la selladora industrial en cada una de las líneas de la planta.

como pendiente. Finalmente, de forma similar a las señales de Disponibilidad, en caso de una inserción correcta, se comprueban los registros pendientes y, si existen, se realizará la inserción correspondiente en la tabla de datos de Sellado. En caso contrario, el proceso finalizará.

C. Señales y conexiones

En la Figura 5, se muestra el esquema de conexiones de los relés, tanto con la selladora, como con la Raspberry Pi. Como se puede observar, los relés poseen dos partes diferenciadas de actuación: (i) por un lado, las conexiones 13 y 14 que corresponden a los contactos de activación del relé y que reciben la señal procedente de la máquina (20 V), y (ii) los contactos con números del 1 al 12 (con una menor tensión, 3.3 V, que es la tensión de trabajo de los pines GPIO de la Raspberry Pi), y que actúan en función de la señal recibida por los bornes anteriores (13 y 14). Cabe destacar la posible configuración de los contactos, siendo las conexiones 1, 2, 3 y 4 dedicadas a una configuración normalmente cerrada, y las conexiones 5, 6, 7 y 8 para una configuración normalmente abierta.

La señal de disponibilidad, enviada por la selladora industrial, la reciben las conexiones 13 y 14 del primer relé, actuando en el contacto 9 para enviar la señal a la Raspberry Pi mediante el GPIO 03. Mientras, la señal de sellado, de la misma forma, llega al segundo relé a través de sus conexiones 13 y 14, pero en este caso actúa en el contacto 8 y envía la señal a la Raspberry Pi a través de su GPIO 02. Los contactos 5 y 12, del primer y del segundo relé, respectivamente, se utilizan para alimentar el relé.

IV. Resultados y Validación

En esta sección, vamos a comentar en detalle los datos que se obtienen al utilizar nuestro sistema. En concreto, existen diferentes datos obtenidos a través de las mediciones, que van a ser tratados por el sistema, con el fin de monitorizar y almacenar los parámetros que intervienen en el cálculo del OEE. Estos datos son los siguientes:

1. Código de Línea de Producción: este código formado por cuatro dígitos, permite identificar la línea de producción a la que pertenece la máquina de sellado de la cual se obtienen los datos. De esta manera, se identifica de forma unívoca los datos de eficiencia obtenidos para cada línea.
2. *Timestamp*: sirve para determinar el momento exacto en el que se ha producido el evento registrado. Más en detalle, se refiere a la cantidad de segundos transcurridos desde las 00:00:00 UTC del 1 de enero de 1970, hasta el momento en el que se produce dicho evento. En nuestro sistema, utilizamos el *Timestamp* para poder determinar cuándo se ha producido cada sellado en cada línea de producción, y cuándo está disponible cada una de las selladoras.
3. Tipo: este dato se utiliza únicamente para medir la disponibilidad de la máquina. En concreto, identificamos el estado de la máquina con un 1 o un 0, en función de si la máquina está disponible o no, de forma que al combinar dicha información con el *Timestamp*, podremos calcular fácilmente el ratio de disponibilidad en cada momento.

La Figura 6 presenta un ejemplo de las señales relativas a la disponibilidad recogidas con sus correspondientes *Timestamp*. Como se aprecia, podemos ver de forma sencilla cuándo ha estado disponible la máquina en cada momento.

Por otra parte, las Tablas I y II incluyen algunos registros, a modo de ejemplo, de los datos recogidos en la planta, relativos tanto a la disponibilidad como al sellado. Más en detalle, la Tabla I muestra el momento exacto en el que se han producido las operaciones de sellado para las 6 líneas de producción (4001-4006). La primera columna indica el código de línea al que pertenece el sellado y en la segunda columna aparece el Timestamp de dicha operación. Por otra parte, la Tabla II muestra los datos necesarios para determinar la disponibilidad del sistema. La primera columna indica el código de línea, la segunda columna el Timestamp de cuándo ha variado

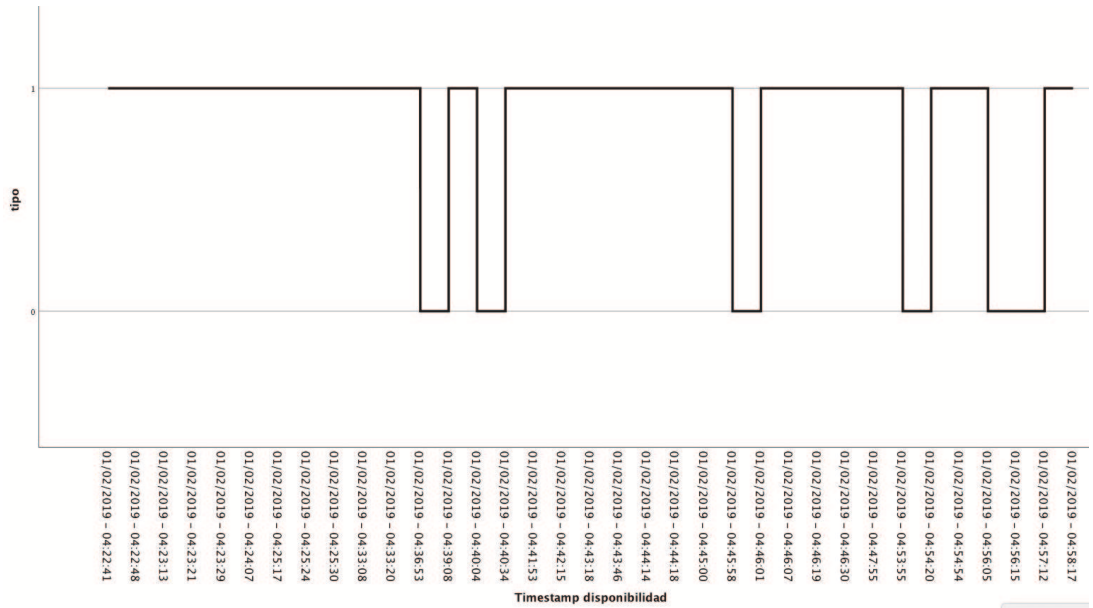


Fig. 6. Gráfico de disponibilidad

TABLA I
Datos de Sellado

Línea	Timestamp
4004	1548996653297
4005	1548996659752
4001	1548996666061
4002	1548996672370
4004	1548996678681
4006	1548996684992
4005	1548996691301
4001	1548996697611
4003	1548996703922

TABLA II
Datos de Disponibilidad

Línea	Timestamp	Tipo
4004	1548995124051	1
4004	1548995129606	0
4004	1548995588100	1
4006	1548995600172	1
4004	1548995812593	0
4005	1548995948159	1
4006	1548996004220	0
4005	1548996033515	1
4003	1548996139873	1

el estado, y en la tercera columna aparece el dato de disponibilidad (0 indica que la máquina no está disponible y 1 que la máquina puede trabajar).

Con estos datos se determinan los tiempos efectivos de trabajo de la máquina, el número de operaciones realizadas en los tiempos programados, tiempos perdidos, pérdidas de velocidad y otros parámetros que son necesarios para el cálculo del índice OEE.

El sistema propuesto, tras el procesamiento de las señales, es capaz de calcular los ratios de Rendimiento y de Disponibilidad, puesto que puede procesar a partir del Tiempo de Producción Planificado, el tiempo perdido debido a la inactividad y a las paradas menores. Además, nuestro sistema permite introducir los datos relativos a la producción efectiva, es decir, el número de paquetes que realmente han sido sellados de forma correcta, con lo que es sencillo estimar el tiempo perdido debido a los productos defectuosos (es decir, lo que denominamos tiempo perdido por defectos).

A modo de resumen, la Tabla III incluye los datos recogidos por nuestro sistema durante 10 horas de trabajo programadas. En concreto, se muestran la horas relativas a las diversas pérdidas (es decir, por paros programados, por inactividad, de velocidad debidas a paradas menores y por defectos), que permiten calcular los diferentes parámetros que afectan a la disponibilidad, al rendimiento y a la calidad. Estos valores permiten obtener el valor final del OEE. La Figura 7 muestra de forma gráfica dichos valores. Finalmente, la Ecuación 2 muestra cómo se ha calculado el OEE a partir de los datos que aparecen en el ejemplo.

$$OEE = 0{,}8824 \cdot 0{,}9333 \cdot 0{,}8143 = 67{,}06\,\% \qquad (2)$$

A la vista de los resultados, los gestores de la empresa podrían observar que para mejorar su productividad sería necesario mejorar sobre todo en los aspectos relativos a la calidad (reduciendo el número de paquetes defectuosos) y de disponibilidad (reduciendo en la medida de lo posible los periodos de inactividad).

V. Conclusiones

En los entornos industriales, la estimación de lo que conocemos como índice de la efectividad global de la maquinaria industrial (*Overall Equipment Effectiveness*, (*OEE*)) permite a los responsables de una empresa detectar qué aspectos deben estudiar en detalle para poder mejorar la productividad y así

TABLA III
DATOS PARA EL CÁLCULO DEL OEE

Concepto	Horas
Horas programadas	10
Horas perdidas por paros planificados	1.5
Horas perdidas por inactividad	1
Horas perdidas por paros menores	0.5
Horas perdidas por defectos	1.3
Tiempo de Producción Planif. (TPP)	8.5
Tiempo Operativo (TO)	7.5
Tiempo Operativo Neto (TON)	7.5
Tiempo Operativo Real (TOR)	7
Tiempo Productivo Neto (TPN)	7
Tiempo Productivo Real (TPR)	5.7

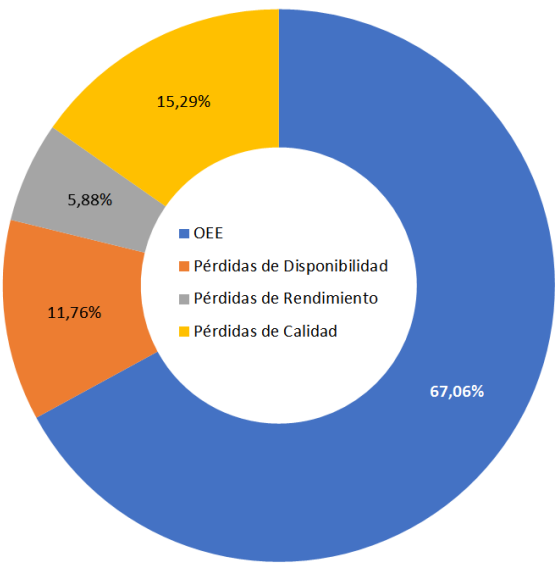


Fig. 7. Gráfica de Eficiencia Global de la Máquina OEE

incrementar la rentabilidad de su empresa. Sin embargo, para poder realizar una estimación del OEE suelen utilizarse autómatas o dispositivos de control propietarios que permitan recoger los datos relativos a la producción, y estos dispositivos requieren una cierta inversión que a veces impide su implantación.

En el presente trabajo, proponemos un sistema basado en dispositivos de bajo coste que permite el cálculo del OEE, además de que los datos obtenidos puedan ser consultados en tiempo real y de forma remota. Nuestra solución permite que se puedan procesar los datos de diferentes líneas de forma paralela y distribuida, lo que hace que nuestro sistema sea altamente escalable y robusto, puesto que el fallo en alguno de los dispositivos desplegados no comprometería la recogida de datos y su tratamiento en el resto de líneas.

Los resultados obtenidos demuestran que el sistema propuesto es robusto y que las mediciones son precisas, habiendo reduciendo notablemente el coste necesario, especialmente si lo comparamos con otras soluciones similares basadas en dispositivos propietarios.

AGRADECIMIENTOS

Este trabajo ha sido parcialmente financiado por el Gobierno de Aragón (Referencia Grupo T40_17D) y el Fondo Social Europeo 2014-2020 Çonstruyendo Europa desde Aragón", así como por el Ministerio de Ciencia, Innovación y Universidades, Programa Estatal de Investigación, Desarrollo e Innovación Orientada a los Retos de la Sociedad, Proyectos I+D+I 2018, Spain (RTI2018-096384-B-I00).

REFERENCIAS

[1] Carlos A. Garcia, Jose E. Naranjo, Tatiana P. Zambrano, and David Lanas y Marcelo V. Garcia, "Low-cost cyber-physical production systems for industrial control robots under IEC 61499," in *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*, Sep. 2018, vol. 1, pp. 1281–1284.

[2] Fundación Telefónica, "La transformación digital de la industria española. informe preliminar," *Ind. Conectada 4.0*, 2015.

[3] Mario Cruz, Pablo Oliete, Christian. Morales, Carlos González, Bruno Cendón, and Alberto Hernández, "Las tecnologías IoT dentro de la Industria Conectada 4.0," *Gobierno de España, Ministerio de Industria, Energía y Turismo, Escuela de Organización Industrial (EOI). Libro digital en: http://a.eoi.es/industria4*, 2015.

[4] Poder Industrial, "https://poderindustrial.com/mexico-requiere-de-cuatro-cuatro-pilares-para-avanzar-a-la-industria-4-0/," 2018, Último acceso 20 de Mayo 2019.

[5] Eduardo Beltrán de Nanclares, "Estándares para la industria conectada, https://www.industriaconectada40.gob.es/sitecollectiondocuments/grupos-trabajo/gt-estandarizacion/estandarizacion-manuf4-0mond-industria-conectada.pdf," 2017, Último acceso 20 de Mayo 2019.

[6] María Guadalupe Morán Solano, Marcelo Vladimir García Sánchez, and Federico Pérez González, "Control de una planta industrial utilizando sistemas de bajo coste," *Ideas en Ciencia*, pp. 19–31, 2015.

[7] PiFace, "http://www.piface.org.uk," 2013, Último acceso 17 de Mayo 2019.

[8] Codesys, "http://www.codesys.com," 2019, Último acceso 17 de Mayo 2019.

[9] Gustavo Caiza and Marcelo García, "Implementación de sistemas distribuidos de bajo costo bajo norma IEC-61499, en la estación de clasificación y manipulación del mps 500," *Ingenius. Revista de Ciencia y Tecnología*, , no. 18, pp. 40–46, 2017.

[10] Álex Patricio Toapanta Guacapiña, *Sistema de monitorización para la industria 4.0. Un enfoque basado en sistemas ciberfísicos*, Ph.D. thesis, ETSI_Diseno, 2018.

[11] Oscar May Tzuc, Renan Quijano Cetina, Juliana González Quijano, and Daniel López, "Sistema de monitoreo inalámbrico de bajo costo para módulos fotovoltaicos empleando raspberry pi," *Pistas Educativas*, vol. 120, pp. 819–839, 10 2016.

[12] Abel Andrés Párraga Román y Jorge Washington Vega Sánchez, "Implementación de un sistema de administración energética mediante Raspberry Pi 3, bajo las condiciones de la norma ISO 50001 aplicado a cargas domésticas," M.S. thesis, Universidad Politécnica Salesiana, 2017.

[13] David Martinez Simarro, "Food industry 4.0. ¿qué supone la digitalización de la industria alimentaria?, https://www.ainia.es/tecnoalimentalia/tecnologia/food-industry-4-0-digitalizacion-industria-alimentaria/," 2016, Último acceso 21 de Mayo 2019.

[14] Seiichi Nakajima, *Introduction to TPM: total productive maintenance*, Productivity Press, 1988.

[15] Robert M Williamson, "Using overall equipment effectiveness: the metric and the measures," *Strategic Work System, Inc*, pp. 1–6, 2006.

[16] Mahsa Fekri Sari and Soroush Avakh Darestani, "Fuzzy overall equipment effectiveness and line performance measurement using artificial neural network," *Journal of Quality in Maintenance Engineering*, vol. 25, no. 2, pp. 340–354, 2019.

Implementación de un algoritmo de filtrado de terreno a partir de datos LiDAR sobre SoC Zynq

Álvaro Vázquez Álvarez[1] , Jorge Martínez Sánchez, David López Vilariño, Francisco Fernández Rivera, José Carlos Cabaleiro y Tomás Fernández Pena[2]

Resumen— **En este artículo se propone la implementación de un algoritmo de filtrado de terreno sobre nubes de puntos LiDAR. Esta etapa de procesamiento es crucial para la extracción de un modelo digital del terreno, lo que a su vez es fundamental en numerosas aplicaciones en el ámbito de la Geoinformática. Se trata de un algoritmo con capacidad para el procesamiento de puntos LiDAR en tiempo de adquisición, dado que solo se precisa disponer de un conjunto reducido de datos: la línea de escaneado a computar y las inmediatamente anterior y posterior. La implementación se ha llevado a cabo en una plataforma híbrida, el kit de desarrollo Zedboard que incluye el SoC Zynq XC7Z020 de Xilinx, que combina en el mismo circuito integrado un procesador de propósito general ARM con lógica reconfigurable (FPGA). Los resultados muestran que es posible portar el algoritmo completo en sistemas SWaP, destinados a aplicaciones donde el tamaño, peso y consumo de energía son limitaciones fundamentales.**

Palabras clave— **Codiseño software/hardware, arquitectura Zynq, procesamiento en tiempo real, aceleradores hardware.**

I. Introducción

La tecnología LiDAR (del Inglés, Light Detection And Ranging) proporciona una fuente de información precisa para numerosas aplicaciones como la clasificación de estructuras terrestres, modelado tridimensional de entornos urbanos, registro de cambios forestales, etc. Los sensores LiDAR permiten conocer la distancia al objeto observado mediante la emisión de pulsos de luz y las medidas del tiempo que les lleva retornar al sensor. Las medidas de distancia resultantes se ajustan mediante referencias terrestres dando lugar a nubes de puntos tridimensionales no uniformes de alta resolución [1].

La mayor parte de las aplicaciones actuales toman como punto de partida lo que se conoce como modelo digital de elevaciones, cuya generación depende en gran medida de la identificación precisa de puntos de terreno (e.g., libre de construcciones o vegetación alta) [2], [3]. Esta etapa de procesamiento, denominada filtrado de terreno, actúa directamente sobre la nube de puntos LiDAR que representa un volumen de datos muy elevado con una distribución espacial habitualmente irregular. El resultado de esta operación es clave no solo para garantizar la precisión del modelo digital del terreno sino también para aliviar la carga computacional de etapas de procesamiento posteriores [4].

Habitualmente los métodos de filtrado de terreno operan sobre los puntos LiDAR asumiendo la disponibilidad de la nube completa lo cual, de facto, representa un procesamiento offline. Comúnmente en LiDAR aerotransportado los datos se adquieren mediante un barrido en líneas perpendiculares a la ruta seguida por el aeroplano que porta el sensor. Por lo tanto, el uso de estas líneas de escaneado como entrada de la etapa de filtrado de terreno resulta razonable para aplicaciones con necesidades de procesamiento en tiempo real o en tiempo de vuelo. En este sentido, hemos desarrollado un algoritmo de filtrado de terreno consistente en un etiquetado unidimensional y bidireccional de los puntos LiDAR en función de la altura y la pendiente así como una interpolación iterativa basada en splines. Este algoritmo es preciso en la extracción de puntos de terreno, siendo además apropiado para su implementación en plataformas de reducido tamaño, peso y consumo de energía cumpliendo con los requerimientos de procesamiento en tiempo real atendiendo a las características de los sistemas de adquisición de datos LiDAR con restricciones similares. Este es el caso, por ejemplo, de aplicaciones en el ámbito de los vehículos aéreos no tripulados (UAV).

En la siguiente sección describimos brevemente el algoritmo de filtrado de terreno desarrollado. A continuación detallamos la implementación del algoritmo sobre un sistema basado en el SoC Zynq de Xilinx así como una evaluación comparativa del rendimiento de una implementación completa en CPU y la alternativa con aceleradores integrados en la lógica programable.

II. Algoritmo de filtrado de terreno

El algoritmo de filtrado de terreno toma como datos de entrada conjuntos de líneas de escaneado y asigna a cada punto una etiqueta en función de si son puntos de terreno o no. La primera etapa consiste en la identificación de vecinos de cada punto que no forman parte de la misma línea de escaneado. A continuación se realiza un proceso iterativo de interpolación mediante splines sobre cada línea de escaneado. Finalmente cada punto es etiquetado como de terreno o no en función de su residual respecto al spline final. A continuación se describen brevemente las etapas principales del algoritmo.

A. Cálculo de vecinos

Dentro de la misma línea de escaneado la identificación de vecinos es inmediata atendiendo al momento de la adquisición. Sin embargo, para identificar los puntos vecinos fuera de la misma línea, esta información resulta insuficiente. Estos vecinos serían el punto más próximo en la línea de escaneado anterior y el punto más próximo de la línea siguiente. La búsqueda de estos vecinos

[1]Dpto. de Electrónica y Computación, Universidade de Santiago de Compostela, e-mail: alvaro.vazquez@usc.es.

[2]Centro Singular de Investigación en Tecnoloxías Intelixentes (CiTIUS), Universidade de Santiago de Compostela, e-mail: jorge.martinez@usc.es, david.vilarino@usc.es, ff.rivera@usc.es, jc.cabaleiro.usc.es, tf.pena.

se realiza partiendo del punto con el mismo índice en la línea vecina y recorriendo dicha línea en un sentido u otro siempre que se reduzca la distancia (figura 1). Es decir, dado un punto p_i y asumiendo p'_i como punto con el mismo índice en la línea adyacente, el procedimiento sería:

Algoritmo 1 Búsqueda de vecinos en líneas adyacentes

Input: punto p_i, línea adyacente de longitud n
Output: vecinos en líneas adyacentes pn

```
pn = p'_i
D_min = distancia(p_i, p'_i)
j = i
while j < n do
    j = j + 1
    D = distancia(p_i, p'_j)
    if D < D_min then
        pn = p'_j
        D_min = D
    else
        exit while
    end if
end while
j = i
while j > 0 do
    j = j - 1
    D = distancia(p_i, p'_j)
    if D < D_min then
        pn = p'_j
        D_min = D
    else
        exit while
    end if
end while
```

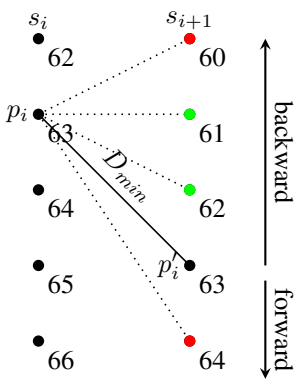


Fig. 1.Búsqueda de vecinos para el punto con índice 63. Los puntos verdes son puntos cuya distancia a p_i es menor que el anterior por lo que se les considera candidatos a vecino. Los puntos rojos representan incrementos en la distancia por lo que se descartan y la búsqueda de vecinos en esa dirección se da por finalizada.

B. Etapa de interpolación

Para realizar la interpolación se ha seleccionado el algoritmo de Akima [5] incluido en la librería GSL [6]. El primer paso es selecionar los vértices o semillas iniciales. El número de semillas depende del algoritmo de interpolación. En el caso del algoritmo de Akima se requieren al menos 5 semillas. La selección de dichas semillas se realiza del siguiente modo: la línea de escaneado se divide en cinco tramos de igual longitud y en cada uno de ellos se selecciona como semilla el punto de menor altura. Inicialmente asumimos que las semillas son puntos de terreno. Esta suposición es razonable siempre y cuando la longitud de la línea sea lo suficientemente grande entendiendo como tal que tenga una mayor longitud que los objetos presentes en la escena. Este criterio es importante por lo que debe tenerse en cuenta en la etapa de adquisición de datos.

Una vez que se dispone de las semillas se procede a la interpolación. El algoritmo utilizado consiste en una interpolación en sub-splines continuamente diferenciables construida mediante polinomios de tercer orden [5]. Una característica importante de este algoritmo es su localidad: los valores de la función en el intervalo $[x_i, x_{i+1}]$ dependen exclusivamente de su vencindad $f_{i-2}, f_{i-1}, f_i, f_{i+1}, f_{i+2}, f_{i+3}$ haciendo que sea un método de interpolación muy eficiente desde el punto de vista computacional dada la alta localidad en los accesos a los datos requeridos.

Dado que se trata de una interpolación bidimensional los puntos LiDAR tridimensionales se proyectan en dos dimensiones del siguiente modo: Las coordenadas originales $\{x, y, z\}$ se transforman en $\{x', y'\}$ donde x' representa la distancia Euclídea (2D) al primer punto de la línea, e y' la altura (i.e., $y' = z$). A continuación se lleva a cabo la interpolación sobre el conjunto de vértices o nudos $K = \{(x'_i, y'_i) : 0, 1 \dots n\}$ resultantes de la proyección de las semillas tridimensionales. Probablemente, el resultado inicial de la interpolación sea una estimación burda del terreno de modo que será preciso un refinamiento mediante la inclusión de vértices adicionales. El reclutamiento de estos nuevos vértices se aborda mediante dos etapas iterativas de repulsión y atracción del spline:

- Etapa de atracción: En esta etapa se hace descender el spline añadiendo puntos de terreno al conjunto K. El punto con mayor residual bajo el spline entre dos vértices consecutivos (k_i, k_{i+1}) se añade al conjunto. solo se admiten puntos cuyo residual supere un umbral previamente determinado. Después de procesar todos los pares de vértices, si el conjunto K tiene más puntos que inicialmente, se actualiza la interpolación. Este proceso se repite hasta que el número de puntos de K se mantiene constante.
- Etapa de repulsión: En esta etapa el spline se eleva hasta los límites de los objetos de la escena añadiendo al conjunto K vértices por encima del spline. La idea principal es partir de un punto de terreno conocido e ir etiquetando como puntos de terreno los puntos sucesivos mientras la transición sea suficientemente suave.

Para integrar información bidimensional se procede a una propagación de información desde la línea previamente procesada a la siguiente. Esta propagación consiste en añadir como vértices de control para la interpolación iterativa de la siguiente línea aquellos puntos vecinos de los vértices resultantes de la interpolación de la línea previa cuya diferencias de altura y pendiente no superen unos umbrales (determinados heurísticamente). Esta estrategia de propagación de información permite una

identificación precisa de los puntos de terreno sin renunciar al procesamiento de datos en streaming.

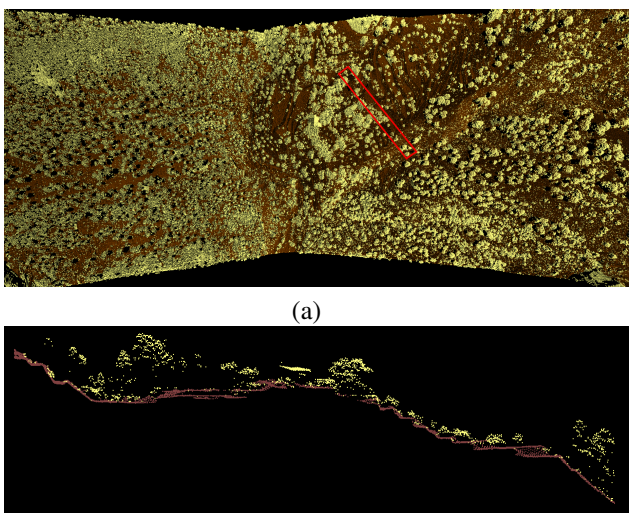
El etiquetado final de puntos como terreno o no terreno se realiza mediante un filtro del residual de cada punto en relación al spline resultante. solo los puntos con residual inferior a un umbral son etiquetados como terreno.

III. Resultados experimentales

El algoritmo ha sido evaluado sobre nubes de puntos correspondientes a diferentes entornos:

- Escenario rural que incluye zonas con pendientes elevadas, cambios de altura del terreno abruptos y presencia de colinas (Alcoy).
- Escenario rural con zonas de vegetación densas (Trabada).
- Escenario urbano (Vaihingen) [7].

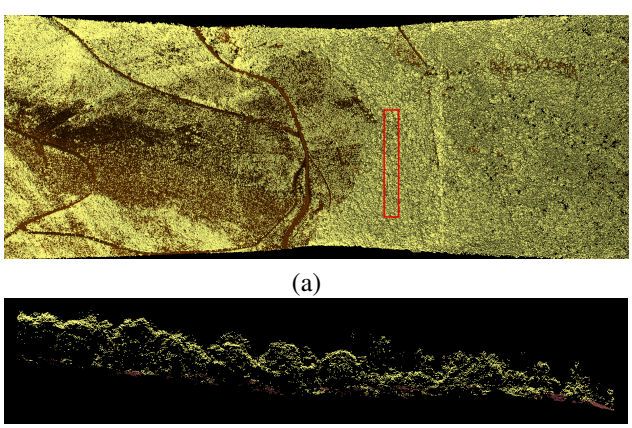
En las figuras 2-4 se muestran los resultados del procesamiento de una línea de escaneado. Para los tres casos, los umbrales de pendiente y diferencia de altura se han fijado en 60° y $0{,}5\ m$ respectivamente. En estas figuras, el color marrón indica que se trata de puntos de terreno mientras que el amarillo indica puntos que no son terreno.



(a)

(b)

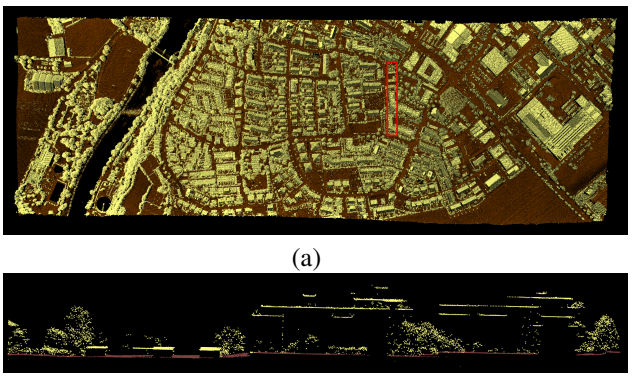
Fig. 2.Nube de puntos *Alcoy*: (a) franja clasificada. (b) perfil de la zona.



(a)

(b)

Fig. 3.Nube de puntos *Trabada*: (a) franja clasificada. (b) perfil de la zona.



(a)

(b)

Fig. 4.Nube de puntos *Vaihingen*: (a) franja clasificada. (b) perfil de la zona.

IV. Implementación en SoC Zynq

El algoritmo desarrollado es apropiado para su implementación en plataformas SWaP que permitan su integración con los sistemas de sensado con vistas a la adquisición y procesamiento de los datos LiDAR en tiempo real. En este sentido, hemos abordado la implementación del algoritmo de filtrado de terreno en sistemas Zynq de Xilinx, tanto a nivel de procesador como de lógica programable, para el desarrollo de aceleradores en la etapa de extracción de vecinos. Concretamente hemos llevado a cabo la implementación sobre la plataforma Zedboard, de bajo coste y bajo consumo de energía y que incorpora un SoC Zynq-7000 de Xilinx y 512 MB de memoria RAM DDR3 [8]. El SoC Zynq incluye una CPU ARM Cortex-A9 de doble núcleo y área de lógica programable para la integración de aceleradores hardware sencillos [9]. En este trabajo hemos abordado dos implementaciones diferentes del algoritmo sobre la plataforma Zedboard:

1. La primera implementación consiste en una migración directa del código desarrollado para su ejecución en equipos de sobremesa (arquitectura x86-64 de Intel o similares).
2. En la segunda implementación introducimos un acelerador hardware sobre la FPGA dedicado a la etapa de extracción de vecinos.

A continuación incluimos una descripción detallada de las dos implementaciones así como una comparativa entre ambas.

A. Implementación en CPU

Una implementación ”bare metal”, es decir, sin soporte de sistema operativo, del algoritmo sobre el SoC Zynq no proporciona suficiente flexibilidad para futuros desarrollos y actualizaciones. Por este motivo hemos optado por integrar una distribución de Linux de 32 bits en el ARM Cortex A9. Concretamente, hemos seleccionado el Kernel de Linux de Xilinx 4.14.04 y el sistema de archivos de root de Ubuntu 16.04. Al portar el código C original al ARM hemos tratado de evitar cambios sustanciales en el mismo. Por este motivo, hemos utilizado librerías precompiladas (en particular GSL que incluye el algoritmo de interpolación de Akima) y soporte para punto flotante del ARM. Del mismo modo, hemos mantenido las directivas OpenMP presentes en el código original. De este

Operación	Zedboard Tiempo (s)	PC Tiempo (s)
Lectura	58,17	3,13
Vecinos	122,90	8,92
Filtrado	48,47	2,58
Escritura	88,37	6,54
TOTAL	317,91	21,17
Rendimiento sin I/O (10^6 pts/min)	1,29	19,28
Rendimiento total (10^6 pts/min)	0,70	10,47

TABLA I

Tiempos de ejecución para *Vaihingen* (3776182 puntos LiDAR).

Operación	Zedboard Tiempo (s)	PC Tiempo (s)
Lectura	120,31	10,04
Vecinos	36,53	1,82
Filtrado	55,52	4,24
Escritura	142,12	15,79
TOTAL	354,49	31,89
Rendimiento sin I/O (10^6 pts/min)	3,91	59,40
Rendimiento total (10^6 pts/min)	1,02	11,29

TABLA II

Tiempos de ejecución para *Trabada* (6000000 puntos LiDAR).

modo, la implementación del algoritmo en el ARM es directa. Por contra, al tratarse de un algoritmo diseñado originariamente para CPUs tipo Intel x86-64, esta implementación directa sobre el ARM de la Zynq aunque funcional, no está optimizada desde el punto de vista del rendimiento. En las tablas I, II y III se muestra una comparativa de tiempos de ejecución sobre las nubes de puntos de referencia (Alcoy, Trabada y Vaihingen) para la implementación directa en la Zedboard (ARM Cortex-A9 de doble núcleo a 667 MHz, sin aceleradores hardware) y el PC de referencia (Intel Core i7-4790 y 16 GB de memoria RAM, sistema operativo Linux Ubuntu 14.04.5 LTS). Como era de esperar, el rendimiento de este último es claramente superior al observado en la Zedboard. En cualquier caso, el rendmiento de la Zedboard es suficiente para la mayoría de los casos prácticos atendiendo al tiempo de adquisición de los datos a procesar.

B. Aceleradores en lógica programable

Con vistas a mejorar el rendimiento de la implementación del algoritmo de filtrado de terreno sobre la Zynq explotando los recursos de lógica programable disponibles, hemos desarrollado un acelerador hardware para el cálculo de vecinos. Esta operación es más costosa en entornos urbanos que en rurales debido a la presencia de más structuras verticales que afectan a la densidad local de puntos. En la figura 5 se muestra la arquitectura del sistema implementado.

El acelerador consta de unidades para la búsqueda de vecinos (denominadas NCOREs en la figura 5), que precomputan los puntos vecinos de diferentes líneas de escaneado simultáneamente. Cada módulo NCORE se comunica con la RAM directamente a través de los módulos DMA y los puertos HP disponibles. Para una utilización óptima del ancho de banda de memoria se requieren dos controladores DMA de 64 bits (módulos AXI-DMA en la figura 5) para cada NCORE implementado. Esta restricción en el ancho de banda solo afecta a la lectura de datos, dado que las operaciones de escritura en RAM son menos frecuentes. La frecuencia de reloj de la lógica programable es de 100 MHz mientras que la de la CPU es de 667 MHz. Los bloques DMA admiten una frecuencia máxima de 150 MHz.

Operación	Zedboard Tiempo (s)	PC Tiempo (s)
Lectura	118,15	9,64
Vecinos	12,38	0,71
Filtrado	74,78	6,11
Escritura	159,96	17,37
TOTAL	365,27	33,83
Rendimiento sin I/O (10^6 pts/min)	4,54	58,02
Rendimiento total (10^6 pts/min)	1,08	11,71

TABLA III

Tiempos de ejecución para *Alcoy* (6600000 puntos LiDAR).

En la figura 6 se muestra un diagrama general del sistema software-hardware desarrollado. La interacción del acelerador hardware para el cálculo de vecinos con el resto del algoritmo de filtrado de terreno, que se ejecuta en el procesador ARM bajo el sistema Linux, se gestiona mediante las herramientas y controladores de Xilinx para los DMA.

La arquitectura de los NCOREs se muestra en la figura 7. Además del paralelismo a nivel de línea de escaneado (una línea por NCORE), cada NCORE puede realizar la búsqueda de vecinos de modo concurrente para varios puntos LiDAR consecutivos dentro de la misma línea, asignando una búsqueda a cada SPU (Stream Processing Unit) implementada en el NCORE. En consecuencia, el paralelismo total implementado en el acelerador hardware viene dado por el número de NCOREs multiplicado por el número de SPUs en cada NCORE.

El número óptimo de NCOREs se puede estimar a partir de la relación entre el ancho de banda de memoria disponible y el rendimiento de los NCOREs [1]. En el caso de la Zedboard este factor es próximo a dos. Para la prueba de concepto se ha implementado un módulo con dos SPUs (NCORE 1) y otro módulo con tres SPUs (NCORE 2). Por lo tanto el número total de SPUs disponibles es de cinco. El uso de los recursos de lógica programable se detallan en la tabla IV.

Componente	BRAM18K	DSP48E	FF	LUT
NCORE 1	0	24	5259	7725
NCORE 2	0	36	7513	11088
Infraestructura HW	20	0	17484	11532
Total HW	20	60	30256	30345
Disponible	280	220	106400	53200
Utilizado[%]	7	27	28	57

TABLA IV

Recursos de lógica programable utilizados.

Los recursos de flipflops (FF) se destinan a la implementación de registros. Las LUT se usan para la implementación de lógica combinacional mientras que los

[1]El rendimiento del NCORE es directamente proporcional a la frecuencia de reloj e inversamente proporcional al intervalo máximo entre dos datos de entrada consecutivos.

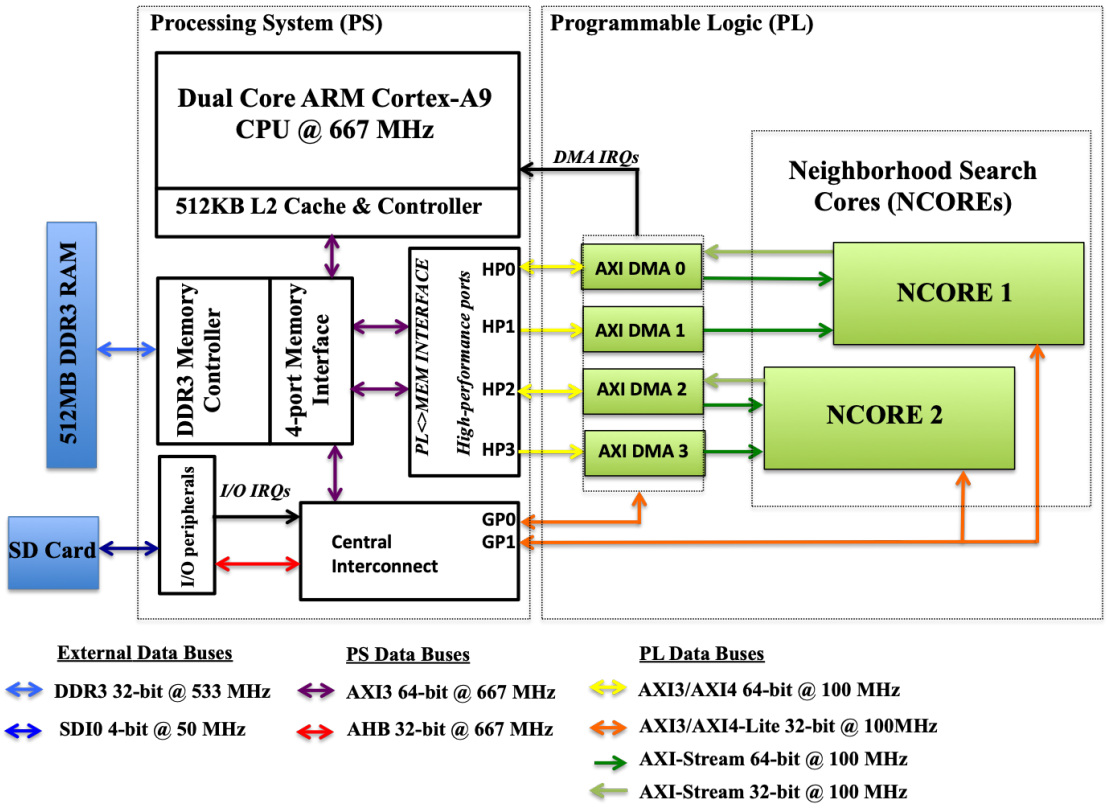


Fig. 5. Implementación del acelerador en el SoC Zynq.

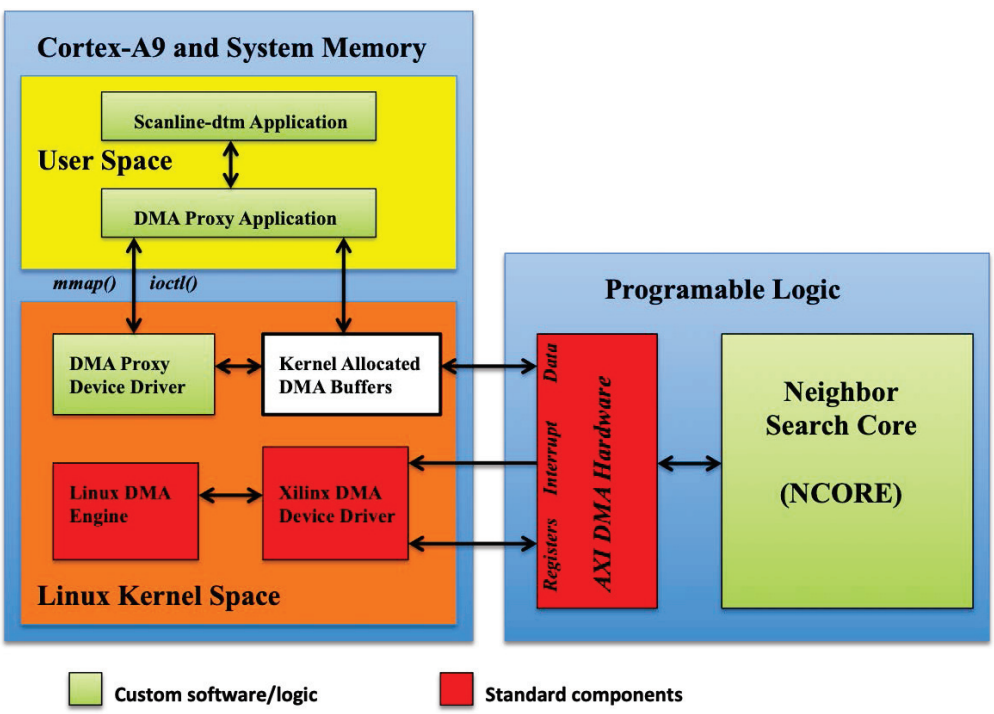


Fig. 6. Arquitectura del sistema implementado.

DSP48E se emplean para operaciones aritméticas, tanto en punto fijo como en punto flotante. La principal limitación en rendimiento es la disponibilidad de LUTs, dado que la integración de más unidades SPU en los NCORE no repercute en un mayor uso del ancho de banda de memoria.

Con el objetivo de proporcionar una estimación de la aceleración obtenida en el procesamiento de la búsqueda de vecinos dentro del algoritmo de filtrado de terreno, mostramos en la tabla V los tiempos de ejecución de una nube de puntos (Vaihingen) en la Zedboard con y sin aceleración. Como puede observarse la búsqueda de vecinos se acelera en un factor $3{,}4\times$ en la versión con acelerador hardware respecto a la implementación directa.

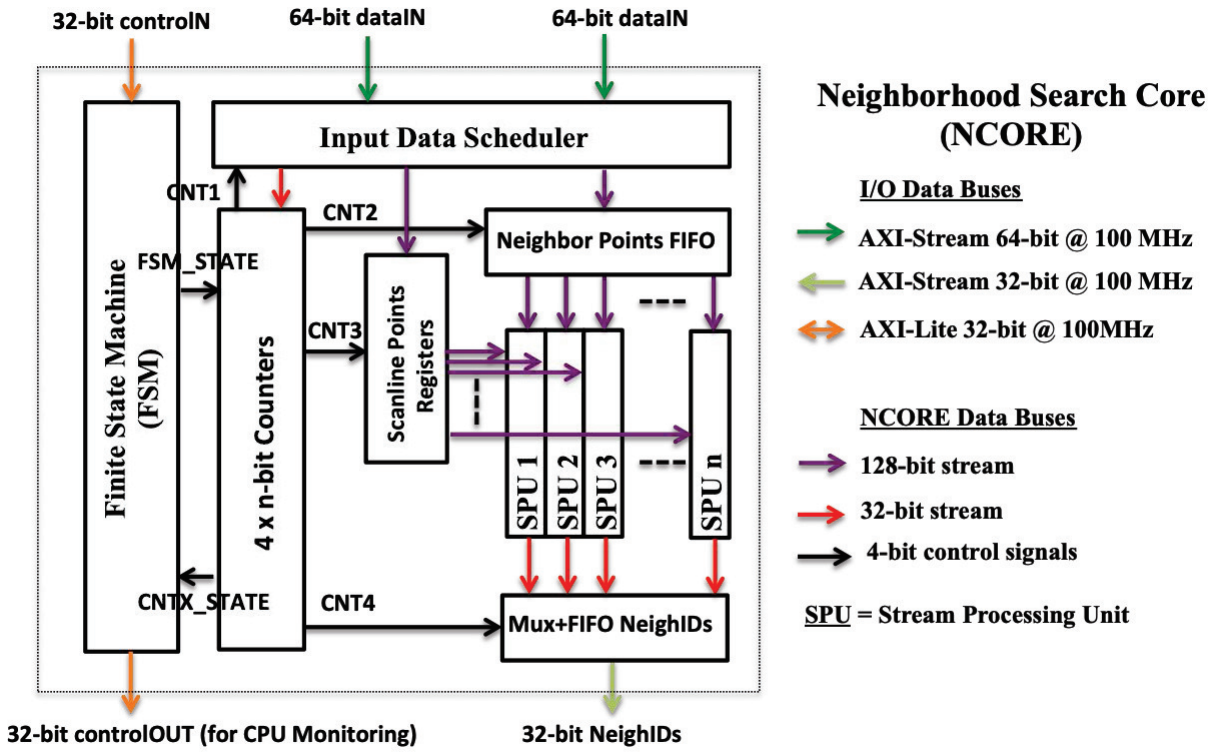


Fig. 7. Arquitectura de las unidades de procesamiento de vecinos (NCORE).

	CPU+Acel HW	CPU
Comp. Vecinos (s)	36	123
Comp. Total (s)	233	318
Rendimiento sin I/O (10^6 pts/min)	2,64	1,29
Rendimiento total (10^6 pts/min)	0,97	0,70

TABLA V
Tiempos de ejecución para *Vaihingen* en la Zedboard

C. Estimación de consumo de energía

El consumo de energía es un aspecto importante en sistemas empotrados orientados a la integración con los módulos de adquisición de datos. Este es el caso de los sistemas LiDAR montados sobre UAVs donde las disponibilidades de espacio, carga y suministro de energía son habitualmente escasas y limitan las características de los sistemas que se pueden embarcar.

Para la estimación de consumo de energía hemos empleado dos métodos:

- Para una estimación preliminar del consumo del SoC Zynq de Xilinx durante la fase de diseño hemos empleado el estimador de consumo de Xilinx (XPE, Xilinx Power Estimator). El XPE es una herramienta que proporciona información detallada del consumo de energía en función de los recursos empleados en el diseño y las frecuencias de reloj.
- Para la estimación del consumo de energía de la Zedboard durante la ejecución medimos la caída de tensión en una resistencia de derivación incluida en la fuente de alimentación de la Zedboard.

La resistencia de derivación, de $10\ m\Omega$, está conectada en serie con la línea de alimentación de la placa por lo que puede usarse para medir la corriente suministrada y por lo tanto el consumo de energía para una tensión de alimentación de $12\ V$.

$$P = \frac{V_{medida}}{10\ m\Omega} 12\ V \quad (1)$$

La disipación de potencia durante los tests realizados tanto con como sin acelerador hardware fue inferior a $3\ W$. En la tabla VI se muestra una comparativa del rendimiento y consumo de energía en el PC de referencia ($130\ W\ TDP$) y las dos implementaciones realizadas en la Zedboard para la nube de puntos de Vaihingen. Como se puede observar, una de las principales fortalezas del uso de la Zedboard es el bajo consumo de energía.

Sistema	Tiempo Proc. [s]	Energía Cons. [J]
PC	21	2730
Zynq (CPU)	318	954
Zynq (CPU+acel HW)	233	699

TABLA VI
Tiempos de ejecución y consumo de energía para *Vaihingen*.

V. Conclusiones

En este trabajo se presenta la integración de un algoritmo para la extracción de puntos de terreno a partir de datos LiDAR sobre un sistema basado en el SoC Zynq de Xilinx. Este algoritmo es altamente preciso en el filtrado de terreno y cumple los requisitos para el procesamiento en tiempo real, proporcionando además una solución de bajo consumo de energía apropiada para ser integrada con las etapas de adquisición en sistemas con restricciones en peso y consumo de energía como es el caso de

los vehículos aéreos no tripulados. Se propone además la aceleración de una parte del algoritmo, la computación de vecinos, explotando los recursos de lógica programable de la Zynq. El diseño propuesto es escalable y fácilmente portable a plataformas de gama más alta dentro de la familia de SoCs Zynq. Tomando como ejemplo el SoC Zynq Ultrascale Zcu104, la disponibilidad de un mayor ancho de banda ($4\times$ frente a la Zedboard) así como un procesador más avanzado (ARM de cuatro núcleos y 1,5 GHz) permitirá acelerar hasta $4\times$ la versión de CPU implementada en la Zedboard. Por otra parte, la disponibilidad de más recursos de lógica programable permitirán integrar un mayor número de módulos para la computación de vecinos, llevando a mejoras en el rendimiento del acelerador entorno a $5\times$ frente a la versión implementada en la Zedboard. A nivel de aplicación el siguiente paso será el desarrollo de una versión propia para la etapa de interpolación que sea adecuada para su integración total o parcial en la lógica programable.

AGRADECIMIENTOS

Este trabajo ha recibido financiación por parte de la Consellería de Cultura, Educación e Ordenación Universitaria de Galicia (referencias ED431G/08 y ED431C 2018/19) y el Fondo Europeo de Desarrollo Regional (FEDER). Esta investigación ha sido también financiada por el Ministerio de Educación, Cultura y Deporte (referencia TIN2016-76373-P). Las nubes de puntos LiDAR de Trabada y Alcoy fueron proporcionadas por Babcok International Group.

REFERENCIAS

[1] Jie Shan and Charles K Toth, *Topographic laser ranging and scanning: principles and processing*, CRC press, 2008.

[2] Jie Shan and Sampath Aparajithan, "Urban dem generation from raw lidar data," *Photogrammetric Engineering & Remote Sensing*, vol. 71, no. 2, pp. 217–226, 2005.

[3] Xiaoye Liu, "Airborne lidar for dem generation: some critical issues," *Progress in Physical Geography: Earth and Environment*, vol. 32, no. 1, pp. 31–49, 2008.

[4] Xuelian Meng, Nate Currit, and Kaiguang Zhao, "Ground filtering algorithms for airborne lidar data: A review of critical issues," *Remote Sensing*, vol. 2, no. 3, pp. 833–860, 2010.

[5] Hiroshi Akima, "A new method of interpolation and smooth curve fitting based on local procedures," *J. ACM*, vol. 17, no. 4, pp. 589–602, 1970.

[6] J. Theiler B. Gough G. Jungman M. Booth M. Galassi, J. Davies and F. Rossi, *GNU scientific library reference manual*, 2002.

[7] M. Cramer, "The DGPF-test on digital airborne camera evaluation - overview and test design," *Photogrammetrie-FernerkundungGeoinformation*, vol. 2, no. 8/W2, pp. 73 – 82, 2010.

[8] Louise H Crockett, Ross A Elliot, Martin A Enderwitz, and Robert W Stewart, *The Zynq Book: Embedded Processing with the Arm Cortex-A9 on the Xilinx Zynq-7000 All Programmable Soc*, Strathclyde Academic Media, 2014.

[9] Xilinx, "Xilinx Zynq-7000 SoC Technical Reference Manual. UG585 (v1.12.2) July 1, 2018.," https://www.xilinx.com/support/documentation/user_guides/ug585-Zynq-7000-TRM.pdf, 2018.

Uso de Redes Neuronales en Procedimientos de Aproximación Final de Aeronaves

Guillermo Tomás Fernández Martín, Pablo Olivas Auñón,
Aurelio Bermúdez Marín, Rafael Casado González [1]

Resumen— **Dado el incremento del tráfico aéreo en los últimos años, que pronostica un crecimiento aún más pronunciado en el futuro cercano, se hace cada vez más necesario un sistema eficiente de automatización total o parcial del control de dicho tráfico. Con el auge de la computación mediante GPU y las redes neuronales, este trabajo plantea la posibilidad de diseñar e implementar un sistema que sustituya total o parcialmente la función de control aéreo en la fase de aproximación final al aeropuerto, mediante un sistema híbrido de redes neuronales y lógica tradicional.**

Palabras clave— **Redes neuronales, simulación, espacio aéreo, gestión del tráfico aéreo, aproximación final.**

I. Introducción

El imparable incremento a nivel mundial en el número de operaciones aéreas [1] hace necesario introducir mecanismos que lleven a cabo una gestión eficiente del espacio aéreo, y en especial del tráfico en el entorno aeroportuario.

En este trabajo se explora la posibilidad de emplear redes neuronales [2] durante las maniobras de aproximación final a pista de las aeronaves. El objetivo es que las redes sirvan de apoyo a la función de control de tráfico aéreo (ATC, *Air Traffic Control*), ya que actualmente es la torre de control del aeropuerto la que se encarga de dirigir estas operaciones.

Por un lado, se contemplará el empleo de la red neuronal en el caso (más frecuente) en el que la aproximación se realiza sin ningún tipo de contratiempo, y por tanto de acuerdo al plan de vuelo establecido en la carta de navegación correspondiente. También se considerará un escenario en el que la maniobra de aproximación deba abortarse como consecuencia de alguna situación inesperada, como por ejemplo una climatología adversa. Este segundo escenario, denominado aproximación frustrada (o *missed approach*) es menos común, pero introduce una sobrecarga importante en el tiempo de vuelo, con el consiguiente incremento en el consumo de combustible, ruido en el entorno, etc., por no hablar de que puede afectar al nivel de satisfacción de los viajeros.

El entorno de trabajo de partida para esta línea de investigación está basado en una herramienta de simulación de espacio aéreo. Dicha herramienta modela el comportamiento de la aeronave y de la torre durante la aproximación a pista, y es la base sobre la que se están desarrollando las primeras propuestas para la mejora de los procedimientos de aproximación a pista.

La principal aportación de este trabajo consiste en detallar una de las formas en las que las redes neuronales están siendo incorporadas a este entorno de trabajo, proporcionar algunos resultados preliminares y esbozar las líneas de trabajo futuras.

El resto del trabajo se estructura de la siguiente forma. En primer lugar, la Seccion II introduce los mecanismos de navegación aérea, centrándose en el procedimiento de aproximación, y aporta algunos trabajos relacionados. A continuación, la Sección III describe el entorno de modelado empleado y presenta una primera mejora relacionada con la maniobra de aproximación frustrada. Después, la Sección IV describe el empleo de redes neuronales durante el procedimiento de aproximación habitual y la Sección V detalla el uso de redes en caso de aproximación frustrada. Finalmente, la Sección VI proporciona algunas conclusiones preliminares y describe el trabajo futuro a desarrollar.

II. Antecedentes y Estado del Arte

A. Navegación Aérea

Los sistemas de navegación aérea han evolucionado notablemente en las últimas décadas [3]. Inicialmente las rutas aéreas se establecían en base a un conjunto de radiobalizas (conocidas como VOR, *VHF Omnidirectional Range*) ubicadas en posiciones fijas en tierra, de forma que la navegación de la aeronave consistía en pasar de una radiobaliza a la siguiente. Posteriormente se evolucionó al sistema RNAV (*aRea NAVigation*), que proporciona navegación directa hacia el destino, a través de un conjunto de puntos de ruta (o *waypoints*) definidos mediante un nombre, una latitud y una longitud, conectados mediante líneas geodésicas y almacenados en una base de datos para mayor comodidad del piloto.

El sistema de navegación más novedoso es la navegación basada en prestaciones (PBN, *Performance-Based Navigation*) [4], en el que las rutas incorporan trayectorias acotadas e incluso curvas, y donde la tripulación de la aeronave es alertada si se exceden ciertos límites de error respecto a la ruta definida. La navegación PBN está fuertemente soportada por diversas tecnologías, tales como el sistema de localización de alta precisión GBAS (*Ground-Based Augmentation System*), y los sistemas de comunicaciones ADS-B (*Automatic Dependant Surveillance - Broadcast*) y CPDLC (*Controller-Pilot Data Link Communications*).

[1]Dpto. de Sistemas Informáticos, Instituto de Investigación en Informática de Albacete, Univ. de Castilla-La Mancha, e-mail: guillermotomas.fernandez@alu.uclm.es, pablo.olivas@alu.uclm.es, aurelio.bermudez@uclm,es, rafael.casado@uclm.es.

B. Aproximación Final y Maniobra Frustrada

La aproximación [5] es una fase crítica de vuelo. Existen procedimientos de aproximación visual, donde el piloto acepta toda la responsabilidad del aterrizaje, y procedimientos de aproximación instrumental (IAP, *Instrument Approach Procedure*), donde el piloto se limita a seguir la carta de aproximación a pista. Los segundos están compuestos de una serie de segmentos (inicial, intermedio, final y missed approach), cuyo comienzo y final se establecen mediante una serie de localizaciones o fijos de aproximación. El primero de estos fijos es el IAF (*Inital Approach Fix*).

Una vez completados los segmentos inicial y final, la aeronave alcanza el segmento final de la aproximación, donde dispone de una serie de ayudas de navegación para llegar a la pista de aterrizaje. Cuando la aeronave dispone únicamente de guiado lateral (2D), proporcionado, por ejemplo, por una estación VOR, se habla de aproximación final de no precision. Sin embargo, durante una aproximación final de precisión, la aeronave cuenta con guiado lateral y vertical (3D), proporcionado normalmente por un ILS (*Instrument Landing System*). En la actualidad el ILS está siendo sustituido progresivamente por el GLS (*GBAS Landing System*), un sistema basado en el mencionado sistema GBAS.

La maniobra frustrada es el procedimiento que se sigue cuando, por cualquier motivo, el piloto decide que la aeronave no puede tomar tierra. En estas situaciones, una vez alcanzado el MAPt (*Missed Approach point*) indicado en la carta de aproximación, la aeronave deberá ejecutar las instrucciones indicadas en la propia carta o las proporcionadas por el ATC. En el mejor de los casos, la aeronave se dirigirá a un fijo alejado del tráfico y, desde allí, se encaminará al IAF para iniciar de nuevo la maniobra. No obstante, en muchas ocasiones la aeronave debe permanecer en vuelo, a la espera de nuevas instrucciones del ATC.

C. Trabajos Relacionados

En la literatura pueden encontrarse numerosos trabajos relacionados con la mejora de la gestión del espacio aéreo y, más concretamente, de los procedimientos de aproximación. Además, la mayoría de las propuestas se evalúan mediante técnicas de modelado y simulación del espacio aéreo. En [6] se proporciona una perspectiva actualizada de todos estos desarrollos. Algunos de ellos se han llevado a cabo en el marco de proyectos que pretenden flexibilizar las rutas seguidas por las aeronaves. En otros casos, el objetivo del trabajo es reducir el impacto del tráfico aéreo sobre el medioambiente. Hay también trabajos enfocados a evitar obstáculos, colisiones, o zonas de vuelo prohibidas o con condiciones climáticas adversas. Un objetivo recurrente de estos trabajos es la mejora de la capacidad del espacio aéreo y, más específicamente, la capacidad de las pistas de aterrizaje de los aeropuertos. Además, muchos trabajos recientes en esta línea tienen como objetivo la mejora de los procedimientos en el contexto PBN.

Las redes neuronales también se han empleado en el campo de la gestión del espacio o del tráfico aéreo. Por ejemplo, la Administración Nacional de la Aeronáutica y del Espacio (NASA), en colaboración con la Administración Federal de Aviación Norteamericana (FAA), está desarrollando un sistema de prevención de colisiones para aviones no tripulados basado en el uso de redes neuronales profundas [7]. En [8] el objetivo de la red es detectar y resolver posibles conflictos en el entorno del aeropuerto. En [9] y [10] se emplean redes neuronales (y otros clasificadores) para predecir el nivel de carga en el ATC, y en [11] la red se usa como soporte para el ATC a la hora de asignar una pista de aterrizaje. Otros trabajos emplean la red neuronal para clasificar y predecir trayectorias [12][13][14] o anticipar retardos en los vuelos [15][16].

III. Entorno de Trabajo

A. Simulador de Espacio Aéreo

Se ha empleado la herramienta *Matlab/Simulink* de *MathWorks* [17] para implementar un simulador de espacio aéreo, sobre el que se han modelado todos los elementos que intervienen en las maniobras de aproximación y aterrizaje de una aeronave en un aeropuerto. El entorno desarrollado nos permitirá desarrollar y analizar los mecanismos de modificación dinámica de rutas a emplear para evitar posibles conflictos o incrementar la capacidad del espacio aéreo. El diagrama del simulador, desarrollado como un sistema multiagente, se puede observar en la Figura 1

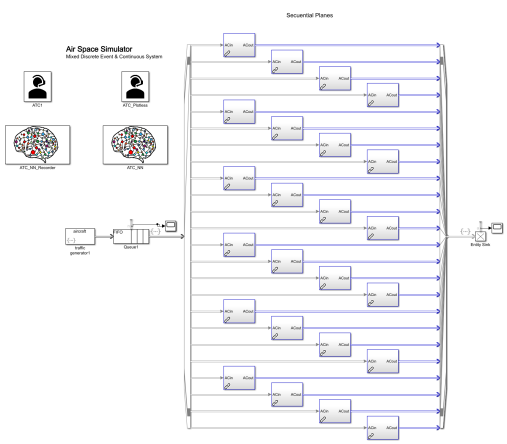


Fig. 1. El simulador de tráfico aéreo modelado en *Simulink*

En el estado actual del simulador, éste es capaz de modelar el aterrizaje de unas aeronaves con las mismas características y dinámicas que el popular Airbus A320 en una diversidad de aeropuertos. Estos aeropuertos vienen definidos por el paradigma actual de aproximación utilizado en aeropuertos reales, es decir, el uso de *waypoints*. Un *waypoint* es una posición en el espacio en tres dimensiones junto con una velocidad a la que la aeronave debe alcanzar dicha

posición.

Además, el simulador permite una representación visual de dichas aeronaves en el espacio tridimensional, gracias al paquete existente en *Simulink* que permite representar y actualizar una serie de puntos en el espacio. Con dicho paquete, se permite pausar la simulación en cualquier momento dado y rotar la gráfica en cualquiera de los tres ejes para observar el recorrido que deberían seguir y la posición real en la que están situadas las aeronaves; pudiendo reanudar en cualquier momento la simulación. Esta visualización, que ralentiza la simulación para una correcta visualización de las aeronaves, se puede deshabilitar en caso de necesitar realizar simulaciones de manera masiva, aumentando enormemente el rendimiento del simulador.

Las dinámicas de los aviones vienen modeladas desde el paquete *Aircraft Dynamics* contenido en *Simulink*, que permite la abstracción de los comportamientos más realistas de las aeronaves para enfocarse de una manera más precisa en las técnicas de aproximación existentes y las posibles mejoras.

El elemento clave del simulador es el ATC, que procesa los mensajes enviados por las aeronaves en vuelo en cada ciclo de simulación. Esta es la forma en la que se modelan los radares de aproximación tradicionales o los modernos sistemas ADS-B. Este envío de mensajes se lleva a cabo con la librería *SimEvents*, también incluida en *Simulink*, que permite la implementación de un sistema basado en eventos de manera sencilla.

Los mensajes recibidos en el ATC contienen la posición en el espacio de cada aeronave, un identificador y un flag que se activa en caso de aterrizaje frustrado. Como resultado del procesamiento llevado a cabo, el ATC debe generar un nuevo *waypoint* para cada aeronave cuando es preciso. Es en este contexto donde se plantea la posibilidad de utilizar una aproximación basada en redes neuronales para el modelado del ATC, de una manera que permite la generación y envío de dichos *waypoints* de una manera transparente al programador.

B. Modelado del Procedimiento de Aproximación

Como se ha detallado en la sección anterior, el procedimiento empleado por el simulador es el mismo que se emplea en la actualidad en la gran mayoría de aeropuertos internacionales: la aproximación mediante *waypoints*. Este procedimiento consiste en la definición de una zona de aterrizaje o aeropuerto por medio de una serie de puntos en el espacio que llevan asociados un identificador. Estos *waypoint* se definen de manera estática, de tal forma que todos los aviones seguirán exactamente la misma ruta en circunstancias normales. Así, la manera de modelar la aeronave consiste en un vuelo constante hacia el *waypoint* que cada avión tenga almacenado, y que sea el ATC el que decida en cada momento cuándo es necesario entregar un nuevo *waypoint* a cada avión. Es este ATC quien contiene la lista estática de *waypoints* que define el aeropuerto, desconocidos en un primer momento para la aeronave. Sin embargo, el momento de entrega de un *waypoint* no es trivial. En el paradigma de aproximación utilizado, el momento de entrega de un *waypoint* no depende sólo de la distancia o de haber alcanzado el *waypoint* anterior, sino también de la velocidad del trayecto. Cuando se cumplen una serie de condiciones, la aeronave salta de un *waypoint* a otro, de tal manera que deja de volar en la línea recta en la que se hallaba para dirigirse hacia el nuevo *waypoint* en línea recta cuanto antes. Esto hace que para las curvas más cerradas a altas velocidades sea preciso entregar el nuevo *waypoint* mucho antes que para los virajes más sutiles a baja velocidad, más cerca de la pista de aterrizaje.

C. Mejora de la Maniobra Frustrada

Usando el procedimiento descrito, se ha propuesto una mejora en la gestión del tráfico aéreo que en la actualidad no se encuentra implementada en ningún aeropuerto. Dicha propuesta está basada en el concepto de operación y la tecnología que da soporte a la navegación basada en prestaciones (PBN) anteriormente comentada. Esta mejora consiste en la reinyección de los aviones que deban frustrar la maniobra de aproximación. Como se ha descrito, en la actualidad, en caso de tener que abortar la maniobra de aproximación, las aeronaves deben dirigirse a una zona de espera alejada del aeropuerto en la cual deben permanecer hasta que la torre de control les permita regresar. En el mejor de los casos, cuando nada más llegar a dicha zona de espera la torre de control da la autorización, las aeronaves gastan una cantidad de combustible y tiempo notoria, además de las posibles indemnizaciones que la compañía aérea debe cubrir por retraso del vuelo.

Con la mejora propuesta, se busca un hueco en el flujo de aterrizaje de las aeronaves que sea lo suficientemente amplio como para poder albergar a la aeronave reinyectada sin ningún tipo de inconveniente para ninguno de los otros dos aviones entre los cuales se reinyecta. El criterio para determinar si dicho hueco existe es que en todo momento se cumplan las separaciones mínimas (tanto en horizontal como en vertical) establecidas en la normativa [18].

Esta maniobra se ha implementado tomando la libertad de manipular la lista estática de *waypoints* que componen un aeropuerto. En el caso de una aproximación frustrada que sea candidata para reinyectar, es decir, que existe un hueco en el cual la aeronave se puede alojar, se generan tres *waypoints* nuevos que se añaden a la lista que contiene el ATC. Estos tres *waypoints*, que varían en cada caso de reinyección (puesto que el hueco donde el avión reinyectará también lo hace), suponen la manera más eficiente de proporcionar una nueva ruta segura a la aeronave. Tras su reinyección en el flujo aéreo, la aeronave se dispondrá a alcanzar el *waypoint* más cercano a su punto de reincorporación. La trayectoria seguida por la aeronave se puede observar en la Figura 2, donde se observa que el hueco existente g en el momento de abortar la maniobra se mueve hasta una posición fu-

tura g_e, que será el hueco donde el avión reinyectará finalmente.

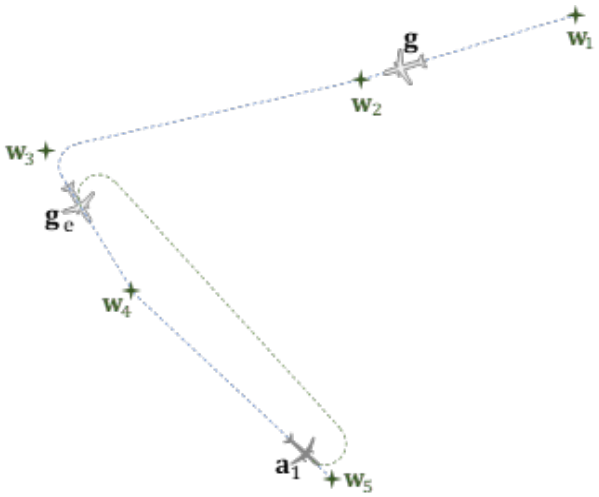


Fig. 2. Trayectoria de reinyección.

IV. Incorporación de Redes Neuronales

Dado el auge que las redes neuronales y las técnicas de *Deep Learning* están teniendo en la actualidad, debido a la gran mejora de las arquitecturas y al uso de las GPU para el entrenamiento eficiente de los distintos algoritmos existentes, se plantea en este trabajo la posibilidad de sustituir la lógica de control del ATC, implementada en el simulador de manera tradicional, por una red neuronal capaz de desempeñar este trabajo.

A. Aproximación con una Red Neuronal y Técnicas de Deep Learning

El planteamiento inicial de este trabajo fue la utilización de una gran red neuronal que aprovechara la potencia de las nuevas GPU existentes en el mercado. Además, dado que el entorno de trabajo y el simulador original es *Matlab/Simulink*, se optó por utilizar el kit de redes neuronales de *Matlab*, que permite una fácil implementación en *Simulink*.

Sin embargo, después de varios intentos fallidos, se optó por abandonar esta propuesta. La gran cantidad de opciones y comportamientos que se recogen en el simulador aumentan exponencialmente la dificultad de aprendizaje, y el kit de *Matlab* empleado en el momento no contenía la potencia necesaria para manejar una red de las dimensiones requeridas. Por ello, se abandonó esta línea de trabajo en favor de la que se explicará a continuación.

B. Aproximación con Múltiples Redes Neuronales

Puesto que el principal problema encontrado era la ingente cantidad de comportamientos distintos que el simulador lleva a cabo en los distintos momentos de su ciclo de vida, el planteamiento adoptado fue localizar y aislar las diferentes tareas que lleva a cabo el simulador y diseñar una pequeña red neuronal para cada una de dichas tareas. Posteriormente las tareas se unirán con una lógica tradicional notablemente más sencilla que la existente.

Las redes neuronales que se utilizan para este trabajo son redes completamente conectadas, que constan de una sola capa de entrada y una capa de salida. Gracias a la división de tareas, estas sencillas redes son capaces de aproximar el trabajo que realiza el simulador en cada una de sus fases. A la hora del entrenamiento, se ha optado por una estrategia de entrenamiento supervisado, dado que podemos obtener los datos necesarios del simulador.

B.1 Aproximación al Aeropuerto

El primer grupo de redes neuronales se dedica íntegramente a modelar la aproximación normal al aeropuerto. Esto consiste en la generación y entrega de *waypoints* a cada uno de los aviones. La primera diferencia que se puede apreciar con respecto a la implementación tradicional es que se ha decidido generar cada uno de los *waypoints*, en lugar de cogerlos de una lista. Esta decisión se ha tomado para reducir el número de elementos en memoria que se deben almacenar, ya que se puede aprovechar la potencia extra que da la red neuronal para generar dichos puntos.

B.2 Diseño del Sistema

La parte de aproximación al aeropuerto consta de dos redes neuronales como se puede observar en la Figura 3, interconectadas mediante una lógica simple. La primera de ellas se encarga de controlar cuándo se genera un nuevo *waypoint*. Debido a que la lógica de conceder un nuevo *waypoint* está integrada en el ATC, en lugar de dentro de la aeronave, en cada ciclo se recibe el estado del avión, siendo el ATC el que debe tomar la decisión de responder a estos mensajes de estado con un *waypoint* cuando sea necesario.

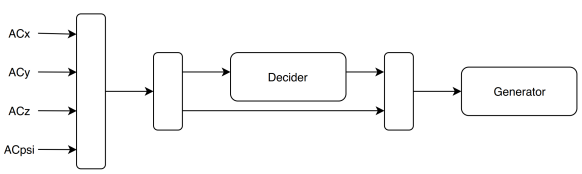


Fig. 3. Esquema del sistema de redes neuronales para la aproximación normal al aeropuerto

Anteriormente se ha comentado que la entrega de un nuevo *waypoint* depende tanto de la posición del avión como del *waypoint* que se quiere alcanzar. Esto nos plantea un inconveniente con respecto a la intención inicial de no almacenar ningún *waypoint*, ya que nos faltan datos para que un algoritmo tradicional sea capaz de dar de manera efectiva un *waypoint* cuando es necesario. Sin embargo, la implementación de la red neuronal se beneficia de dos características existentes en el simulador. Por un lado, el simulador es determinista, de tal manera que los mensajes que se mandan en un ciclo son los mismos en cada ejecución del simulador. Por otro lado, como se ha dejado antes claro, la maniobra de aproximación al aeropuerto se compone de una serie estática de *waypoints*. Por ello, aun con la ligera variación de llegada de cada avión, se van a formar una serie de puntos

calientes en el espacio tridimensional en los cuales aumentará de forma notable la probabilidad de que se conceda un *waypoint* a un avión que se encuentre ahí.

Es así como trabaja la primera red neuronal de este sistema. Como entrada se opta por las tres coordenadas espaciales de un avión en vuelo junto con su orientación, de tal manera que se pasa un vector de cuatro elementos a la red. Como salida de ésta, se obtiene un valor 0 o 1, que indica si se concede *waypoint* o no. En detalle, se optó por la configuración por defecto que *Matlab* proporciona, es decir, una red neuronal con una capa de 10 neuronas y una capa de salida de 1 neurona (tantas como distintas salidas tenga la red).

La segunda red neuronal del conjunto es la que nos generará los *waypoints* que el avión necesita para aterrizar de manera satisfactoria. De la misma manera que la red anterior, ésta se beneficia del hecho que los *waypoints* que definen el aeropuerto sean estáticos. De esta manera, cuando se conceda un *waypoint* en un punto determinado del espacio de coordenadas será el mismo y único para todos los aviones que pasen por ahí. Por esto, se utiliza la misma estrategia que para la red anterior: la entrada de la red la comprenden las coordenadas y orientación del avión, mientras que la salida está formada por los cuatro elementos que definen un *waypoint* que se han mencionado anteriormente (las tres posiciones espaciales y la velocidad que el avión debe tener al alcanzar ese punto).

B.3 Entrenamiento del Sistema

Para entrenar de manera supervisada la red neuronal, la manera más adecuada es contar con una base de datos de registros que contengan todos los casos representativos del comportamiento que queremos que aprenda. Dado que tenemos el código del simulador, podemos modificarlo para obtener distintas bases de datos a partir de lo que suceda en las distintas simulaciones. Cabe destacar que éste es el caso en el que más nos vamos a beneficiar de desactivar la interfaz de visualización del aterrizaje, tal y como se ha mencionado anteriormente.

Para la primera red, se almacenan registros de todos los mensajes que entran al ATC, así como si el ATC ha enviado un mensaje o no. Dado que el kit de redes neuronales de *Matlab* acepta un fichero separado por comas de entrada y otro de salida, se almacenaron los datos en ficheros separados.

Aquí cabe destacar que, al recoger la base de datos de esta manera, sin ningún tipo de filtrado, se obtuvieron resultados muy poco eficientes, debido a la disparidad de los registros. Por la propia naturaleza del simulador y de la red diseñada, al recoger los registros se observó que se almacenaban una cantidad de resultados negativos (que no ofrecían *waypoint*) tan grande que los registros que sí ofrecían *waypoint* era despreciable, lo que causaba un sesgo enorme en el entrenamiento y unos malos resultados. Para solventar este problema, se optó por la eliminación aleatoria de algunos registros negativos, de tal manera que la base de datos final tuviera una cantidad aproximadamente equilibrada de registros positivos y negativos.

El algoritmo de entrenamiento utilizado en la red es el algoritmo de Levenberg-Marquardt [19][20], basado en mínimos cuadrados. De aquellos algoritmos implementados en el kit ofrecido por *Matlab*, resultó el más eficiente en términos de tiempo, así como el más preciso. Es importante destacar que *Matlab* no ofrece una implementación que aproveche la GPU para ninguno de sus algoritmos, a excepción del método del gradiente descendiente estocástico [21]. Sin embargo, este método, el más básico de todos, no ofrecía un rendimiento tan bueno incluso con toda la potencia de cómputo adicional. Por ello, se optó por un entrenamiento más lento pero con mejores resultados por CPU. Si bien es verdad que otras herramientas ofrecen una gran variedad de algoritmos implementados para GPU (Keras [22], PyTorch [23], etc.), la posterior implementación de las redes ya entrenadas en *Matlab* tenía una complejidad mayor del tiempo ganado en el entrenamiento.

Por otro lado, la segunda red se entrena con registros generados a partir de aquellos *waypoints* que se han entregado a los aviones. En este caso, no existe un sesgo claro hacia un lado u otro del problema, sino que el mayor inconveniente es la lentitud en la que se generan los registros. Por ello, la velocidad del simulador y la eliminación de características no imprescindibles para esta parte (como fue la parte encargada de la mejora que contempla la reinyección) fueron cruciales a la hora de obtener una base de datos con la que poder entrenar. En cuanto al algoritmo de entrenamiento, es el mismo que se empleó para la red anterior y que tan buenos resultados ofreció.

Para entrenar ambas redes neuronales, se optó por un procedimiento de *holdout* para realizarlo de manera adecuada. El procedimiento consiste en escoger un 70 % de la base de datos de manera aleatoria para entrenar, usar un 15 % para validar ese entrenamiento y realizar las modificaciones necesarias (*backpropagation*) y el 15 % restante para el test, que no se puede utilizar para modificar el algoritmo y que nos da el resultado más preciso y honesto de lo buena que es la red neuronal.

B.4 Resultados del Sistema

Matlab ofrece dos métricas para evaluar el rendimiento de la red neuronal: el Error Cuadrático Medio (ECM) y la recta de regresión. El ECM es una manera de medir qué error comete el modelo a la hora de predecir los datos, mientras que la recta de regresión R indica la correlación que tienen los datos en el modelo. Cuanto más pequeño sea el error cometido, así como cuanto más se acerque la correlación a 1, mejor es nuestro modelo.

El ECM de ambas redes varía en gran medida, ya que en una se encuentran las coordenadas en miles de metros; mientras que la otra contiene 0 o 1. Por ello, se usa principalmente la correlación indicada por la

recta de regresión para indicar lo bueno que es el modelo.

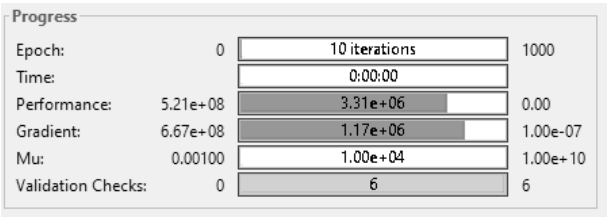


Fig. 4. Estadísticas del entrenamiento proporcionadas por *Matlab*. El campo *Performance* indica el ECM.

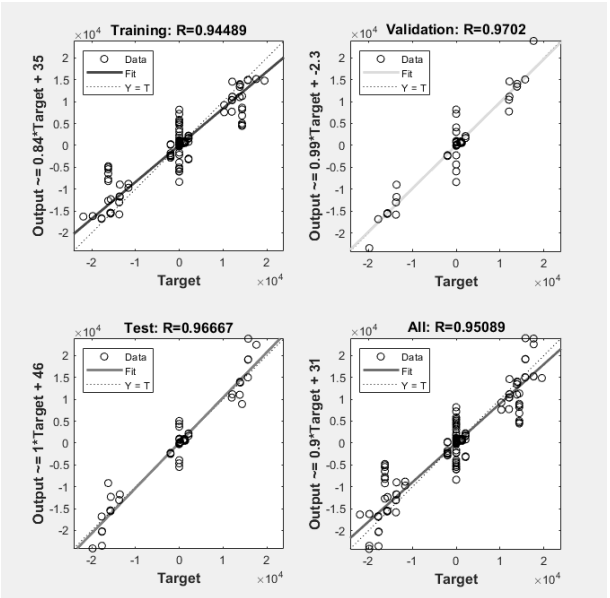


Fig. 5. Recta de regresión para los diferentes conjuntos de la base de datos.

V. Reinyección mediante Redes Neuronales

La segunda parte de este trabajo se centra en implementar con redes neuronales la mejora planteada en la Sección III-C. Debido a la gran cantidad de cálculos matemáticos que se deben realizar para generar los tres *waypoints* necesarios para reinyectar en el tráfico el avión que frustra el aterrizaje, el algoritmo implementado en el simulador se ejecuta de manera demasiado lenta. El objetivo de esta serie de redes neuronales es optimizar y acelerar dicho tiempo de cómputo. Para ello se distinguen dos objetivos clave: la obtención del hueco de reinyección y la generación de *waypoints*. El esquema del sistema de redes neuronales propuesto se puede observar en la Figura 6

A. Hueco de Reinyección

La localización del hueco de reinyección ha sido una de las partes más complejas de este trabajo. Debido a la naturaleza variable de los aviones dentro del simulador, un avión no puede conocer en principio dónde se hallan todos los aviones en todo momento. Por ello, se planteó almacenar en todo momento la posición de todos los aviones, de tal manera que el ATC siempre conozca cuántos aviones se encuentran

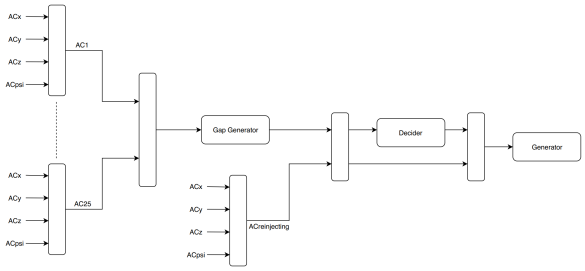


Fig. 6. Esquema propuesto para el aprendizaje e implementación de la reinyección de las aeronaves.

en vuelo y en qué posición están. Una vez se tiene esta estructura de aviones, se planteó la posibilidad de dividir el espacio aéreo en una suerte de pasillo y dividirlo en partes iguales. Así, la entrada a la red neuronal sería un vector binario de los huecos ocupados y los huecos no ocupados y la salida sería en qué posición del pasillo existe un hueco de tamaño suficiente. Sin embargo, la gran cantidad de huecos necesarios en el pasillo para una correcta detección del hueco aumentaban la complejidad de la red neuronal de manera que resultaba inasumible.

Finalmente, se optó por comparar las parejas de aviones usando un sistema de dos redes neuronales. La primera red indica si el hueco existente entre ambos aviones es suficientemente amplico como para que quepa una aeronave más entre ellos, mientras que la segunda red devuelve la posición en la que debería reinyectar. En este caso, las redes tienen distintas entradas. La primera red, que detecta si existe un hueco, recibe las coordenadas de ambos aviones y devuelve un valor 0 o 1. El proceso de recogida de datos y entrenamiento es equivalente al de la red explicada en el punto anterior. Por otro lado, la entrada de la segunda red consiste en la posición del avión trasero y la posición del avión que desea reinyectar. El motivo de que se use sólo un avión es que se puede calcular el punto de reinyección como la distancia a un solo avión, ya que la red neuronal anterior ha definido que existe seguridad de que existe dicho hueco. La utilización del avión que quiere reinyectar se debe a que el punto calculado será el punto en que el avión se colocará en el futuro, con lo cual dependiendo de donde se encuentre el avión en el momento en que decida reinyectar, el punto futuro puede variar.

Los resultados obtenidos con estas redes también fueron satisfactorios tras pulir los problemas, de tal manera que arrojan unos puntos suficientemente precisos para una reinyección sin problemas.

B. Generación de Waypoints

Por último, la generación y entrega de estos *waypoints* comprenden la parte más compleja de este trabajo. Debido a la naturaleza dinámica de estos tres *waypoints*, no se puede aprovechar la pareja de redes que controlan el comportamiento de cualquier avión en circunstancias habituales. El enfoque dado comienza de la misma manera: una red para averi-

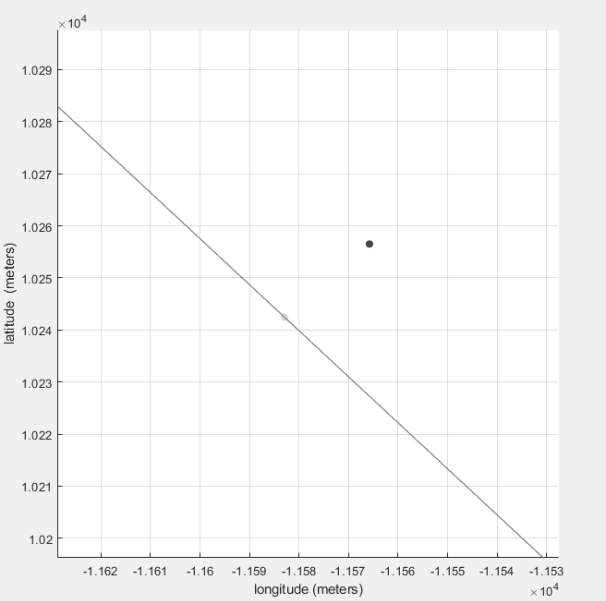


Fig. 7. Sobre la línea, el hueco real. En oscuro, el hueco calculado por la red neuronal.

guar cuándo es el momento de dar el *waypoint* y otra red para generar y entregar dicho *waypoint*.

La red que controla el momento de dar el *waypoint* resulta más complicada que la primera que se realizó. Debido a que los *waypoints* varían en el espacio dependiendo del avión y del hueco, no basta con la sola posición del avión para que esta red sea capaz de saber de manera precisa cuándo debe dar un punto. Aunque se intentó utilizar esta idea, ni siquiera aumentando tanto el tiempo de entrenamiento como la complejidad de la red se consiguieron resultados satisfactorios. Aunque sobre el papel la red aprendía de manera adecuada, consiguiendo un ECM mínimo y una correlación muy alta, a la hora de la implementación se podía observar que no desempeñaba un trabajo adecuado. Esto se debe a que, al no tener conciencia del siguiente *waypoint*, entregaba los dos últimos al mismo tiempo al tener un punto caliente más amplio que el obtenido en la red neuronal del aterrizaje normal.

Para la generación de los *waypoints* se optó por una red cuya entrada fuera la posición del avión y la posición en la que se reinyectará, contando con un total de 7 entradas y 4 salidas. Sin embargo, de igual manera, la alta varianza entre los distintos puntos, junto con las limitaciones encontradas en el simulador a la hora de hacer simulaciones excesivamente largas con una gran cantidad de aviones realizando la maniobra de reinyección, hizo que la red neuronal diera puntos muy impares. Es por esto que se optó por dividir esta red en dos, que generasen el primer y el segundo *waypoint* de manera independiente, y usar el hueco generado anteriormente como tercer *waypoint* de reinyección. Todo este proceso nos obliga a almacenar más datos en la estructura que nos guarda la posición de los aviones, como el estado en que se encuentran de la reinyección (cuántos *waypoints* de reinyección han alcanzado), pero a cambio permite una precisión notablemente mayor. De igual manera, las redes entrenadas resultan mucho más sencillas, constando con la misma configuración de 10 neuronas en la primera capa que las otras.

Para el entrenamiento de estas redes se alterna entre el algoritmo de Levenberg-Marquardt que se ha mencionado anteriormente y el algoritmo de Regularización Bayesiana [24] que implementa *Matlab*. Este segundo algoritmo nos proporciona un resultado mejor en la red que controla el momento de dar un *waypoint* a un avión en reinyección, a costa de un tiempo de entrenamiento y un consumo de memoria más elevado. Como el objetivo final, sin embargo, es la implementación del sistema en un entorno simulado o real, pero con un nivel de realismo y adecuación muy altos, resulta interesante usar el algoritmo que proporcione los mejores resultados finales.

VI. Conclusiones

Este trabajo esboza una primera aproximación para incorporar redes neuronales en un sistema crítico como es la gestión de tráfico aéreo en un aeropuerto. Se puede observar que, utilizando un sistema de múltiples redes unidas entre sí mediante una lógica simple se puede llegar a una aproximación del funcionamiento real del ATC. Sin embargo, también se observa en este estudio que, dada la precisión con la que deben funcionar los sistemas de navegación aérea, la precisión de las redes generadas aún no se encuentra a la altura para reemplazar completamente a los sistemas tradicionales. No debemos olvidar que en este entorno la seguridad es un aspecto crítico.

Con un estudio futuro, y técnicas más complejas, como la inclusión de *Deep Learning* y el uso de herramientas que permitan mayor flexibilidad a la hora de crear y entrenar redes neuronales, se espera un mejor rendimiento del sistema. Otro aspecto que habrá que tener en cuenta necesariamente una vez mejorada la propuesta es el de su certificación, tanto a nivel software como hardware.

Agradecimientos

El presente trabajo ha sido financiado por el Ministerio de Ciencia, Innovación y Universidades del Gobierno de España a través del proyecto RTI2018-098156-B-C52 y por la Universidad de Castilla-La Mancha a través de la ayuda 2019-GRIN-27060.

Referencias

[1] ICAO, "Traffic growth and airline profitability were highlights of air transport in 2016," Tech. Rep. January, 2017.
[2] Jürgen Schmidhuber, "Deep learning in neural networks: An overview," *Neural Networks*, vol. 61, pp. 85–117, jan 2015.
[3] Albert Helfrick, "The centennial of avionics: Our 100-year trek to performance-based navigation," *IEEE Aerospace and Electronic Systems Magazine*, vol. 30, no. 9, pp. 36–45, sep 2015.
[4] ICAO, "Performance-based Navigation (PBN) Manual," Tech. Rep., 2013.
[5] FAA, "Instrument Procedures Handbook (IPH)," 2017.
[6] ZW Zhong, "Overview of recent developments in modelling and simulations for analyses of airspace structures and traffic flows," *Advances in Mechanical Engineering*, vol. 10, no. 2, pp. 168781401775391, feb 2018.

[7] Kyle D. Julian, Mykel J. Kochenderfer, and Michael P. Owen, "Deep Neural Network Compression for Aircraft Collision Avoidance Systems," *Journal of Guidance Control and Dynamics*, 2018.
[8] Ming Qiang Chen, "Flight conflict detection and resolution based on neural network," in *Proceedings - 2011 International Conference on Computational and Information Sciences, ICCIS 2011*, 2011, pp. 860–862.
[9] D. Gianazza, "Learning air traffic controller workload from past sector operations," in *12th USA/Europe Air Traffic Management R and D Seminar*, 2017, pp. 1–7.
[10] Hongyong Wang, Duo Gong, and Ruiying Wen, "Air traffic controllers workload forecasting method based on neural network," in *Proceedings of the 2015 27th Chinese Control and Decision Conference, CCDC 2015*, 2015, pp. 2460–2463.
[11] Yoichi Nakamura, Ryota Mori, Hisae Aoyama, and Hyuntae Jung, "Modeling of runway assignment strategy by human controllers using machine learning," in *AIAA/IEEE Digital Avionics Systems Conference - Proceedings*, 2017, vol. 2017-Septe.
[12] Christabelle S. Bosson and Tasos Nikoleris, "Supervised Learning Applied to Air Traffic Trajectory Classification," in *2018 AIAA Information Systems-AIAA Infotech @ Aerospace*, 2018.
[13] Xiangmin Guan, Renli Lv, Liang Sun, and Yang Liu, "A study of 4D trajectory prediction based on machine deep learning," in *Proceedings of the World Congress on Intelligent Control and Automation (WCICA)*, 2016, vol. 2016-Septe, pp. 24–27.
[14] R. Kaidi, M. Lazaar, and M. Ettaouil, "Neural network apply to predict aircraft trajectory for conflict resolution," in *2014 9th International Conference on Intelligent Systems: Theories and Applications, SITA 2014*, 2014.
[15] Young Jin Kim, Sun Choi, Simon Briceno, and Dimitri Mavris, "A deep learning approach to flight delay prediction," in *AIAA/IEEE Digital Avionics Systems Conference - Proceedings*, 2016, vol. 2016-Decem.
[16] Daniel Alberto Pamplona, Li Weigang, Alexandre Gomes de Barros, Elcio Hideiti Shiguemori, and Claudio Jorge Pinto Alves, "Supervised Neural Network with multilevel input layers for predicting of air traffic delays," in *2018 International Joint Conference on Neural Networks (IJCNN)*, 2018, pp. 1–6.
[17] The MathWorks, Inc, "Matlab product page," https://www.mathworks.com/products/matlab.html, Accessed: 2019-05-22.
[18] ICAO, "Doc 4444. PANS-ATM, Procedures for Air Navigation Services. Air Traffic Management," 2016.
[19] Kenneth Levenberg and Frankford Arsenal, "A Method for the Solution of Certain Non-Linear Problems in Least Squares," *Quarterly of Applied Mathematics*, vol. 1, no. 278, pp. 536–538, 1943.
[20] Manolis Lourakis, "A Brief Description of the Levenberg-Marquardt Algorithm Implemened by levmar," Tech. Rep. January 2005, 2005.
[21] Shai Shalev-Shwartz, Shai Ben-David, Shai Shalev-Shwartz, and Shai Ben-David, "Stochastic Gradient Descent," *Understanding Machine Learning*, pp. 150–166, 2014.
[22] "Keras: The Python Deep Learning library," https://keras.io, Accessed: 2019-05-22.
[23] "PyTorch," https://pytorch.org/, Accessed: 2019-05-22.
[24] David J. C. MacKay, "A Practical Bayesian Framework for Backpropagation Networks," *Neural Computation*, vol. 4, no. 3, pp. 448–472, 1992.

Detección de Colisiones entre Aeronaves mediante Redes Neuronales

Pablo Olivas Auñón, Guillermo Tomás Fernández Martín, Rafael Casado González, Aurelio Bermúdez Marín [1]

Resumen— **En este trabajo se exploran diversas posibilidades a la hora de emplear redes neuronales en una aplicación relacionada con la gestión del tráfico aéreo. El objetivo es que, una vez entrenadas, nuestras redes sean capaces de determinar de manera anticipada si una determinada configuración de aeronaves en fase de aproximación final a una pista de aterrizaje llegarán a quebrantar durante la maniobra los requisitos de separación mínima establecidos en la normativa. El trabajo introduce una herramienta de simulación que ha servido para generar los datos para el entrenamiento de nuestras redes y para validar su comportamiento. Asimismo, detalla los diseños realizados sobre diversas plataformas de desarrollo de redes neuronales (*Keras*, *TensorFlow* y *PyTorch*) y proporciona algunos resultados preliminares.**

Palabras clave— **Redes neuronales, simulación, espacio aéreo, gestión del tráfico aéreo, detección de colisiones.**

I. Introducción

Este trabajo se enmarca dentro una línea de investigación en la que se pretende introducir ciertas mejoras en los mecanismos de gestión de tráfico aéreo. Más concretamente, la línea se centra en la mejora de las operaciones en el entorno aeroportuario, y viene motivada por el incesante incremento del tráfico aéreo a nivel global [1].

Como es evidente, todas las propuestas que se lleven a cabo en el ámbito aéreo deben respetar escrupulosamente la vasta normativa emitida por las múltiples autoridades con competencia en este sector. Entre estas autoridades se encuentra la OACI (Organización de Aviación Civil Internacional), una agencia de las Naciones Unidas, que promueve la seguridad aérea y el desarrollo ordenado de la aviación civil internacional en el mundo entero. La OACI establece estándares y regulaciones necesarias para la seguridad aérea, eficiencia, regulación y también para la protección del medio ambiente. Uno de los muchos requisitos de seguridad que deben cumplir las operaciones aéreas tiene que ver con el mantenimiento de una separación mínima entre las aeronaves que están en vuelo.

El entorno sobre el que se están desarrollando las propuestas de mejora mencionadas arriba consiste en una herramienta para el modelado y la simulación de espacio aéreo, que en la actualidad refleja el comportamiento tanto de las aeronaves como del control de tráfico aéreo (ATC, *Air Traffic Control*) durante las maniobras de aproximación a pista. En este trabajo se propone el empleo de técnicas de *deep learning* y redes neuronales [2] para incorporar a este entorno de trabajo una herramienta que sea capaz de detectar de manera anticipada el quebrantamiento del requisito de separación mínima entre aeronaves.

Actualmente, nos encontramos en un momento en el que las redes neuronales se encuentran en auge, debido principalmente a los grandes avances que se se están haciendo en cuando a la capacidad de cómputo, alcanzando cuotas inimaginables hace años. Gracias a esto y a la proliferación del desarrollo de aceleradores gráficos, que se han demostrado bastante capaces para el proceso de entrenamiento de estas redes, cada vez se plantean más usos para esta tecnología, que requiere de largos y costosos entrenamientos en términos de computación.

Ya que no existe una sola plataforma que facilite al usuario final desarrollar sus propias redes neuronales para la tarea que desee, la cuestión es conocer, dependiendo de las características de cada problema, cuál va a resultar mejor en cuestión de resultados y tiempo necesario para alcanzar los valores deseados. Por ello este estudio trata de esclarecer estas incógnitas partiendo de una base de datos numérica propia que pueda ser alterada. El objetivo será implementar diferentes redes neuronales utilizando varios *frameworks* y analizar comparativamente sus prestaciones, alterando los tiempos que entrenará cada una de ellas y el tamaño de la base de datos utilizada en el entrenamiento, y todo ello sobre un mismo hardware.

En concreto, se emplearán cuatro plataformas diferentes para el desarrollo de las redes neuronales que ofrezcan la funcionalidad anterior: *Matlab* [3], *Keras* [4], *TensorFlow* [5] y *PyTorch* [6]. Se detallará la forma en la que la red ha sido parametrizada en cada caso y se obtendrán una serie de métricas para conocer qué plataforma funciona mejor, con que parámetros y por qué.

El resto del trabajo se estructura de la siguiente forma. En primer lugar, la Sección II menciona algunos trabajos relacionados e introduce las plataformas de desarrollo de redes neuronales consideradas en este estudio y el entorno de simulación empleado. A continuación, la Sección III detalla las implementaciones llevadas a cabo y la Sección IV muestra los primeros resultados obtenidos. Finalmente, la Sección V proporciona algunas conclusiones y esboza el trabajo a desarrollar en esta línea.

[1]Dpto. de Sistemas Informáticos, Instituto de Investigación en Informática de Albacete, Univ. de Castilla-La Mancha, e-mail: pablo.olivas@alu.uclm.es, guillermotomas.fernandez@alu.uclm.es, rafael.casado@uclm,es, aurelio.bermudez@uclm.es.

II. Estado del Arte y Entorno de Trabajo

A. Trabajos Relacionados

En la literatura existen numerosos trabajos en los que se propone emplear redes neuronales para detectar (y a veces, resolver) colisiones (también denominadas "conflictos") entre aeronaves. Por ejemplo, la Administración Nacional de la Aeronáutica y del Espacio (NASA), en colaboración con la Administración Federal de Aviación Norteamericana (FAA), está desarrollando un sistema de prevención de colisiones para aviones no tripulados basado en el uso de redes neuronales profundas [7].

En [8] el objetivo de la red es predecir posibles conflictos entre distintas aeronaves en el entorno de un aeropuerto. El sistema es, en cierto modo, similar al aquí propuesto, ya que permite estimar la posición de las aeronaves con una antelación de hasta 30 segundos, por medio de una red entrenada con datos reales del aeropuerto (aquí usaremos un simulador para ese fin).

De manera parecida, en [9] se propone el uso de una red neuronal con arquitectura MLP (*MultiLayer Perceptron*) que predice la trayectoria de dos aeronaves y evita su colisión.

B. Plataformas Empleadas

Como se ha comentado, para la realización de las redes neuronales se explorará el empleo de cuatro plataformas diferentes: *Matlab*, *Keras*, *TensorFlow* y *PyTorch*. Hay que señalar que *Matlab* es propietaria, mientras que las otras tres alternativas funcionan sobre *Python*.

B.1 Matlab

Matlab es una buena alternativa para empezar, por su simplicidad, siendo la herramienta de más alto nivel que se va a utilizar. Ofrece herramientas tanto visuales como en código para generar y entrenar las redes neuronales. Además, como el propio simulador funciona sobre *Matlab*, hay un nexo común que simplifica el uso de los datos generados por la misma plataforma.

B.2 TensorFlow

TensorFlow es una librería *open-source* diseñada por *Google* para la manipulación de flujos de datos y aprendizaje automático. Esta librería trabaja con lo que han denominado "tensores", que son conjuntos de datos en forma de vectores y matrices. *TensorFlow* está pensado para trabajar con grandes volúmenes de datos, siendo desarrollado para satisfacer la necesidad de *Google* como empresa de disponer de un sistema capaz de construir y entrenar redes neuronales para detectar y descifrar patrones y correlaciones análogos al aprendizaje y razonamiento usados por los humanos.

B.3 Keras

Keras es un *framework open-source* de alto nivel que permite la creación de redes neuronales. Se trata de una librería escrita en *Python* que funciona sobre *TensorFlow*. Está diseñado para que sea *user-friendly*, modular, extensible y sencillo a la hora de modificar las redes neuronales. Esta plataforma destaca por incluir una serie de redes ya entrenadas para facilitar a los investigadores la inclusión de las mismas en sus proyectos rápidamente. En 2017, *TensorFlow* incluyó parte del código de *Keras* en su propio código fuente.

B.4 PyTorch

PyTorch es una librería de código abierto desarrollada por *Facebook*, programada en *Python* y basada en *Torch*. Proporciona dos características de alto nivel: cómputo de grandes cantidades de datos centrados en la aceleración por GPU y creación de redes neuronales.

C. Popularidad de los Distintos Frameworks

En cuanto a la popularidad de los distintos *frameworks*, podemos analizar la de *Keras*, *TensorFlow* y *PyTorch* gracias a la cantidad de contribuciones realizadas por la comunidad en sus respectivos repositorios. *Matlab*, al no tratarse de código abierto, no permite su inclusión en esta comparativa. Con esos datos, se obtiene que el más popular es *Keras*, posiblemente por su simplicidad de uso. Le sigue de cerca *TensorFlow*, llegando a superarlo en algunos momentos, mientras que *PyTorch* siempre ha estado bastante por debajo, aunque en un constante crecimiento que puede alterar estos resultados en un futuro no demasiado lejano (ver Fig. 1).

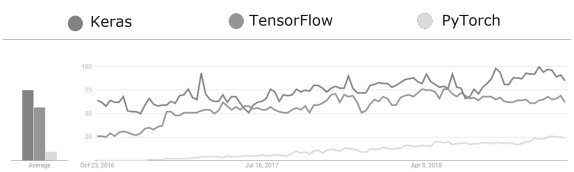


Fig. 1. Popularidad de *Keras*, *TensorFlow* y *PyTorch*

D. Simulador de Tráfico Aéreo

La base de la que se parte es un simulador del espacio aéreo desarrollado sobre la herramienta *Matlab/Simulink*, que modela todos los elementos que intervienen en las maniobras de aproximación y aterrizaje en un aeropuerto, permitiendo además una representación visual de las aeronaves en aproximación en el espacio tridimensional.

El simulador utiliza el paradigma actual de aproximación utilizado en aeropuertos reales, es decir, el uso de *waypoints*. De manera muy resumida, un *waypoint* es una posición en el espacio en tres dimensiones junto con una velocidad a la que la aeronave debe alcanzar dicha posición. Cada pista de aterrizaje dispone de una carta de aproximación en la que aparece toda esta información [10].

Actualmente, el simulador incluye aeronaves con las mismas características y dinámicas que el popular Airbus A320, y modela los aeropuertos de Málaga (ver Fig. 2) y Houston, permitiendo flexibilidad en

el proceso de aparición de nuevos aviones en la zona de aterrizaje y la ruta a seguir de los mismos hasta que estos se encuentran en tierra.

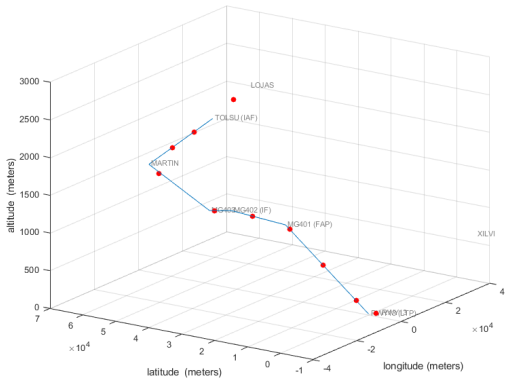


Fig. 2.Varios aviones aproximándose al aeropuerto del Málaga

Uno de los puntos claves del simulador es la figura del ATC (*Air Traffic Controller*), que es el elemento que gobierna todos los movimientos de las aeronaves desde el momento en el que aparecen en las inmediaciones del aeropuerto hasta que tocan tierra. Este ATC es el que, tras la incorporación de las modificaciones que comentaremos seguidamente, detecta las posiciones ilegales de los aviones, avisa de las mismas y crea finalmente la base de datos que posteriormente utilizaremos para entrenar nuestras redes neuronales.

E. Versión Preliminar del Detector

Para la realización de este trabajo se partió del simulador que acabamos de describir, sobre el que se tuvieron que realizar una serie de modificaciones para incluir un detector de posiciones ilegales.

El ATC del simulador utiliza una estructura denominada AC (de *AirCraft*), que contiene información en tiempo de real de todos los aviones; más concretamente, su identificador dentro de esta estructura, su posición y orientación dentro del mapa del simulador, su posición y tiempo de aparición en el radar, así como su velocidad y su estado. Pues bien, aprovechándonos de esta estructura AC, hemos incorporado al ATC una funcionalidad extra que compara la posición de cada uno de los aviones con la del resto y determina si en algún momento dos aeronaves están más próximas de lo que legalmente pueden estarlo. Concretamente, según la normativa vigente, la separación mínima permitida entre dos aeronaves en fase aproximación es de 1000 ft (pies) verticalmente y de 3 NM (millas náuticas) horizontalmente [11].

Es necesario señalar que en una aplicación real esta funcionalidad la llevan a cabo unos sistemas de aviónica embarcados en las aeronaves, denominados TCAS (*Traffic Collision Avoidance System*), y otros sistemas similares ubicados en la torre de control del aeropuerto. Estos últimos emplean la información procedente de los radares de vigilancia u otros sistemas más avanzados (como el ADS-B, *Automatic Dependent Surveillance—Broadcast*) para llevar a cabo estas comprobaciones [12].

III. Implementación del Detector mediante Redes Neuronales

El detector de conflictos incorporado al simulador proporciona la base de datos que utilizaremos para el entrenamiento de nuestras redes neuronales. En concreto, la base de datos consta de dos tablas relacionadas. En la primera se recoge, para cada par de aeronaves, sus posiciones 3D iniciales (x, y, z) y el instante en el que aparecieron en la simulación (es decir, el comienzo de sus respectivas maniobras de aproximación). La segunda tabla contiene para cada registro de la primera un valor booleano que indicará si durante las maniobras de aproximación de ambas aeronaves se ha producido algún conflicto entre ellas.

Por otro lado, para que una red neuronal entrene adecuadamente se necesita que la base de datos sea equilibrada, es decir que contenga aproximadamente el mismo número de resultados de cada uno de los elementos que queremos que pueda predecir en su salida. En nuestro caso, las redes neuronales solo predecirán si hay una colisión (1) o no la hay (0) entre un par de aeronaves. Como en cada simulación hay una mayor proporción de aviones que no están en posiciones ilegales que los sí lo están, solo se ha incluido un tercio de estos pares de aeronaves para balancear la base de datos.

Finalmente, cuando se ha obtenido una base de datos con un tamaño considerable, se comenzará con la creación y entrenamiento de las diferentes redes neuronales, modificando múltiples parámetros para, como se ha indicado, analizar cuál ofrece mejores prestaciones con cada configuración, y por qué.

Las redes neuronales se entrenarán sobre dos tipos de tecnologías diferentes. Se utilizará tanto únicamente el procesador (en este caso un Intel Core i9-9900K, a 3.6GHz) como el procesador y la GPU (una MSI GeForce RTX 2080 Ventus, con 8GB de GDDR6). Es conocido que el entrenamiento de este tipo de redes utilizando GPU es superior al mismo entrenamiento sobre únicamente la CPU; lo que este trabajo trata de analizar es cuánto es mejor el entrenamiento sobre una tecnología que sobre la otra modificando los parámetros que ya fueron indicados.

A. Matlab

Matlab es una buena forma de empezar a trabajar con redes neuronales, al tratarse de la opción que funciona a más alto nivel. Gracias a ello podemos abstraernos de muchas de las características más complejas de las redes, como los algoritmos a utilizar o el número de capas adecuado. Esto que, a priori resulta una ventaja, cuando comienzan a surgir nuevas necesidades para ajustar las redes neuronales a nuestros requisitos, se convierte en un inconveniente. Otra de las desventajas de esta plataforma es que, al ser cerrada y no disponer del código fuente, no se puede conocer exactamente cómo funcionan los algoritmos que utiliza.

El primer paso que se ha llevado a cabo utilizando *Matlab* es la creación de la base de datos a utilizar. La base de datos elegida como entrada será un archivo CSV que contará con ocho columnas. Como se ha comentado, estas columnas indican, para cada par de aeronaves, la posición 3D y el instante en el que las aeronaves aparecieron en la simulación. La base de datos de salida contendrá una sola columna donde indicaremos si hay ilegalidades en sus posiciones o no, marcándolo como un 1 o un 0, respectivamente.

Matlab permite dos formas diferenciadas de trabajar con sus herramientas de aprendizaje automático, ofreciendo una opción más simple mediante una interfaz gráfica y otra algo más compleja y configurable por consola. Ambas, cuando entrenan, muestran la misma ventana gráfica que ofrece información sobre el estado del entrenamiento, tal y como se aprecia en la Fig. 3.

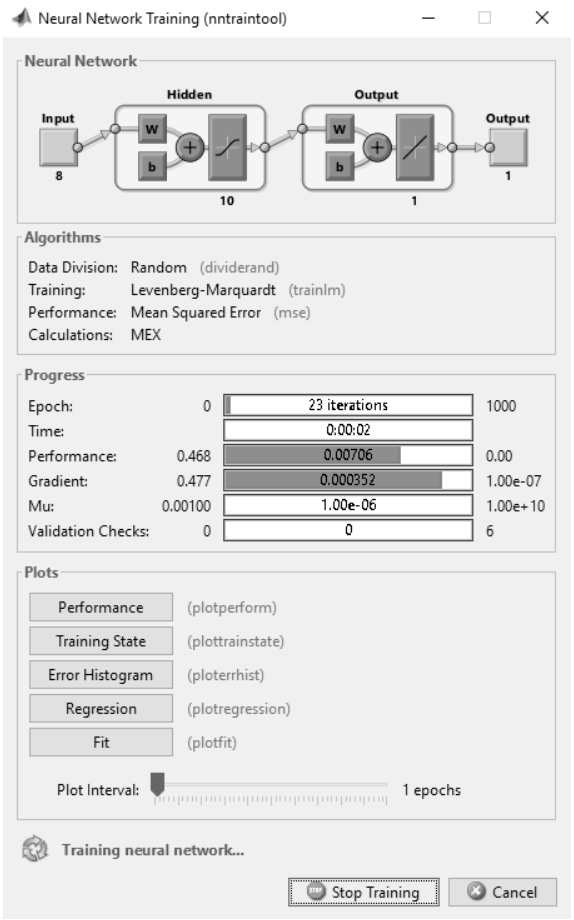


Fig. 3. Proceso de entrenamiento de *Matlab*

Pese a todo, *Matlab* no permite obtener resultados a lo largo del entrenamiento. Esto aumenta considerablemente el tiempo necesario para poder obtener datos, por lo que finalmente se ha decidido descartar el uso de esta herramienta para la creación de redes neuronales, utilizándola únicamente para crear la versión preliminar del detector de conflictos descrito en la sección anterior.

B. Keras

Keras ha sido la segunda opción tenida en cuenta. No funciona a tan alto nivel como *Matlab*, pero sigue siendo más sencilla de utilizar que las otras dos alternativas. Pese a la mayor cantidad de opciones que permite, resulta razonablemente sencillo crear cualquier configuración de red neuronal que se desee.

Al ser la primera opción que se ha escogido para crear una red eligiendo capas libremente, se ha optado por una red pequeña para empezar. En concreto, la red consta de dos capas, la primera con las ocho entradas de la base de datos (detalladas anteriormente) y dieciséis neuronas de salida y *Tanh* como función de activación. La segunda capa contiene una neurona de salida para obtener nuestro resultado y usa la función de activación *Sigmoidal*. Con respecto a la función de pérdida, se ha empleado la *Entropía Cruzada* y como algoritmo de optimización se ha escogido *RMSProp*.

El código utilizado para crear la red neuronal es bastante sencillo, siendo similar al pseudocódigo mostrado en el Listing 1.

Listing 1
Código para *Keras*

```
X = leer_csv(X.csv)
y = leer_csv(y.csv)
X_train, X_test, y_train, y_test
    = obtener_bases_datos(X,y,
    tam_test=0.2)

red{
    capa1 = capa(dim_entrada=8,
        dim_salida=16, Activacion=
        tanh),
    capa2 = capa(dim_entrada=16,
        dim_salida=8, Activacion=
        sigmoid)
    }

red.compilar(optimizador=rmsprop(
    lr=0.001), fun_loss=
    binary_crossentropy, metricas=
    accuracy)
red.entrenar(X_train, y_train,
    epocas=100000, batch_size
    =4096, conj_validacion=0.2)

puntuacion = red.evaluar(X_test,
    y_test, batch_size=4096)
Imprimir(puntuacion)
```

Según se ha visto en otras investigaciones [13], *Keras* resulta mejor que otras alternativas a la hora de trabajar con bases de datos más reducidas, ganando también en cuanto a simplicidad. Nuestro papel ahora es comprobar que esos datos son correctos en nuestro caso, y analizar si alternar entre el uso de

CPU y la GPU puede modificar los resultados.

C. TensorFlow

TensorFlow ha sido la tercera opción elegida. *Keras* funciona sobre ella, por lo que era el siguiente paso lógico para mantener un orden estructurado. En cuanto al tema de simplicidad de uso, complica un poco más el código necesario para generar la redes neuronales que necesitamos para este estudio. La estructura de la red será la misma que en *Keras*, de forma que podamos comparar el rendimiento final con una estructura uniforme.

El código utilizado para *TensorFlow* es algo más complejo, puesto que las clases son algo más abstractas que en *Keras* y utilizan algunos términos diferentes, como "Steps" en lugar de "Epochs". El código quedaría parecido al pseudocódigo mostrado en el Listing 2.

Listing 2
Código para *TensorFlow*

```
X = leer_csv(X.csv)
y = leer_csv(y.csv)
X_train, X_test, y_train, y_test
    = obtener_bases_datos(X,y,
    tam_test=0.2)

red = clasificador(ncolumnas=8,
    activaci nc1=tanh, dim_sal1
    =16, n_clases=2, optimizador=
    RMSPropOptimizer(lr=0.001))

red.entrenar(x=X_train, y=y_train
    , steps=epochToStep(100000),
    batch_size=4096)
perdida = BCELoss()
puntuacion = red.evaluar(x=X_test
    , y=y_test, step=1, batch_size
    =4096)[Accuracy]
imprimir(puntuacion, perdida)
```

Si analizamos estudios anteriores [13] sobre comparativas entre distintos *frameworks*, *TensorFlow* resultaría mejor con grandes bases de datos y ofrecería un gran rendimiento final. Al igual que con *Keras*, se comprobará con nuestra base de datos si estas afirmaciones se cumplen, tanto utilizando CPU como GPU.

D. PyTorch

PyTorch, al ser la plataforma más diferente, se ha dejado para el último lugar. Se trata de un *framework* más a bajo nivel, mucho más complejo de utilizar y que no nos ofrece directamente las funciones de entrenamiento y de pruebas entre otras, por lo que debemos incluir un código que las realice en el propio archivo fuente de la red neuronal. Además, hay que indicarle explícitamente que utilice la GPU pasando todos los datos que queramos mandar a esta y todas las funciones a su propia memoria, por lo que hay que tener un conocimiento más avanzado tanto de redes neuronales como del funcionamiento de estas sobre aceleradores gráficos, puesto que si se realiza inadecuadamente podemos provocar fallos de escasez o ineficacia de la memoria.

Listing 3
Código para *PyTorch*

```
X = leer_csv(X.csv)
y = leer_csv(y.csv)
X_train, X_test, y_train, y_test
    = obtener_bases_datos(X,y,
    tam_test=0.2)

y_train = y_train.Categorias()

X_train = X_train.aCUDA
X_test = X_test.aCUDA
y_train = y_train.aCUDA
y_test = y_test.aCUDA

red = Secuencial(Linear(8,16),
    Tanh(), Linear(16,2), Sigmoid
    ()).aCUDA

fun_perdida = BCELoss().aCUDA
optimizador = RMSProp(lr=0.001)
permutacion = permutacion(len(
    X_train))

for epoca in 0:100000:
    correctos = 0
    imprimir(epoca)
    for i in 0:len(X_train),
        batch_size:
        indices = permutacion(i:i
            +batch_size)
        batch_X, batch_y =
            X_train[indices].aCUDA
            , y_train[indices].
            aCUDA
        prediccion = red(batch_X)
            .aCUDA
        perdida = fun_perdida(
            prediccion, batch_y).
            aCUDA
        correctos += (prediccion
            == batch_y).suma

        optimizador().optimizar
        fun_perdida().optimizar
imprimir(perdida, correctos)
```

El código utilizado para crear la red neuronal utilizando *PyTorch* es similar al pseudocódigo mostrado en el Listing 3. Como se puede observar, *PyTorch* es el único que trabaja con la salida en forma de ca-

tegorías, de forma que nuestra salida tendremos que pasarla a este formato.

Otros análisis [13], enmarcan a *PyTorch* como el mejor en cuanto a flexibilidad, debido al bajo nivel en el que trabaja, al que menos tiempo le cuesta conseguir buenos resultados entrenando y el mejor en cuestiones de *debugging*, facilitando el trabajo al usuario final.

IV. Resultados Preliminares

La Fig. 4 y la Fig. 5 muestran los primeros resultados de rendimiento proporcionados por *TensorFlow*, *Keras* y *PyTorch*, en cuanto al porcentaje de acierto y a la acumulación de error, respectivamente. En concreto, las figuras muestran los resultados ofrecidos por las diferentes redes en función del número de épocas (*Epoch*). Así pues, el resultado final (los valores de la derecha) se obtiene tras entrenar 100,000 veces cada una de las redes con la base de datos de entrenamiento completa (100,000 épocas).

Si nos atenemos únicamente a los resultados ofrecidos por la Fig. 4, parece que, en líneas generales, la plataformas que ofrecen las mejores prestaciones son *TensorFlow* y *Keras*. Si bien *PyTorch* tiene un comportamiento bastante regular a lo largo de todo el entrenamiento, finalmente obtiene resultados inferiores a las otras dos plataformas. Finalmente (con un alto número de épocas) *TensorFlow* obtiene los mejores resultados, estando las tres plataformas por encima de 0.95.

Sin embargo, cuando observamos la acumulación de error (Fig. 5), vemos que las tres plataformas ofrecen resultados muy similares. *Keras* consigue mantener los resultados mas bajos a lo largo de casi todo el entrenamiento, seguido de *TensorFlow*. Finalmente, las tres plataformas obtienen resultados muy parecidos, alrededor del 0.1.

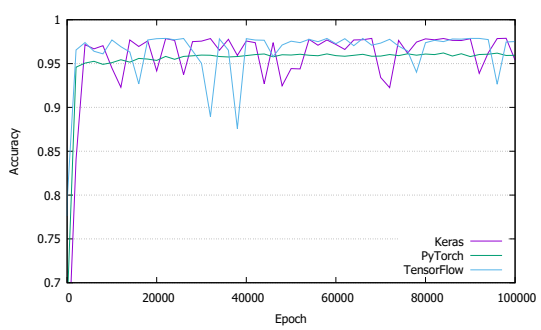


Fig. 4. Comparativa del *Accuracy*

Como se ha comentado, estos resultados son preliminares, obteniéndose de la ejecución de las diferentes redes neuronales utilizando conjuntamente CPU y GPU. El *learning rate* se encuentra situado en 0.001 y el *batch size* en 4,096. La base de datos ronda los 500,000 elementos. Cuando los valores de estos parámetros se alteren, los resultados finales podrían ser distintos, obteniéndose así las diferencias completas en cuanto a funcionamiento de las diferentes plataformas.

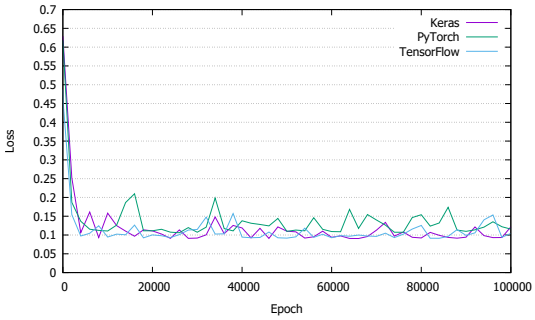


Fig. 5. Comparativa del *Loss*

Nótese que ninguno de estos resultados (independientemente de la plataforma de desarrollo considerada) alcanza un porcentaje de acierto del 100 %. Aunque pudiera parecer que esto es inaceptable en una aplicación tan crítica como la considerada, en realidad esto no es un inconveniente. Actualmente, y hasta donde nosotros sabemos, en la torre de control no existe ningún sistema que detecte de forma anticipada si dos aeronaves llegarán a encontrarse más cerca de lo legalmente permitido, por lo que el sistema aquí descrito permitiría a los controladores anticiparse a dicha situación (aunque no sea en el 100 % de los casos).

V. Conclusiones y Trabajo Futuro

Este trabajo pretende analizar la viabilidad del empleo de redes neuronales para implementar un sistema de apoyo a la gestión del tráfico aéreo que sea capaz de predecir la ocurrencia de colisiones entre un conjunto de aeronaves en fase de aproximación. Se han detallado posibles implementaciones para este detector de colisiones, empleando diversos *frameworks* de desarrollo de redes neuronales, y se han presentado algunos resultados preliminares.

No obstante, como se ha indicado, se hace necesario un análisis mucho más exhaustivo (incluyendo variaciones en el hardware empleado) para poder concluir si las redes neuronales podrían llegar a ser una buena opción a la hora de resolver nuestro problema y, en su caso, cuál sería la plataforma de desarrollo más apropiada.

Agradecimientos

El presente trabajo ha sido financiado por el Ministerio de Ciencia, Innovación y Universidades del Gobierno de España a través del proyecto RTI2018-098156-B-C52 y por la Universidad de Castilla-La Mancha a través de la ayuda 2019-GRIN-27060.

Referencias

[1] International Civil Aviation Organization (ICAO), "Traffic growth and airline profitability were highlights of air transport in 2016," Tech. Rep. January, 2017.

[2] Jürgen Schmidhuber, "Deep learning in neural networks: An overview," *Neural Networks*, vol. 61, pp. 85–117, jan 2015.

[3] The MathWorks, Inc, "Matlab product page," https://www.mathworks.com/products/matlab.html, Accessed: 2019-05-22.

[4] "Keras: The Python Deep Learning library," https://keras.io, Accessed: 2019-05-22.
[5] Google, Inc, "TensorFlow," https://www.tensorflow.org/, Accessed: 2019-05-22.
[6] "PyTorch," https://pytorch.org/, Accessed: 2019-05-22.
[7] Kyle D. Julian, Mykel J. Kochenderfer, and Michael P. Owen, "Deep Neural Network Compression for Aircraft Collision Avoidance Systems," *Journal of Guidance Control and Dynamics*, 2018.
[8] Ming Qiang Chen, "Flight conflict detection and resolution based on neural network," in *Proceedings - 2011 International Conference on Computational and Information Sciences, ICCIS 2011*, 2011, pp. 860–862.
[9] R. Kaidi, M. Lazaar, and M. Ettaouil, "Neural network apply to predict aircraft trajectory for conflict resolution," in *2014 9th International Conference on Intelligent Systems: Theories and Applications, SITA 2014*, 2014.
[10] Federal Aviation Administration (FAA), "Instrument Procedures Handbook (IPH). Chapter 4: Approaches," https://www.faa.gov/, Accessed: 2019-05-23.
[11] International Civil Aviation Organization (ICAO), "Doc 4444. PANS-ATM, Procedures for Air Navigation Services. Air Traffic Management," 2016.
[12] Ian Moir, Allan Seabridge, and Malcolm Jukes, *Civil Avionics Systems*, Wiley, 2013.
[13] "Keras vs TensorFlow vs PyTorch : Comparison of the Deep Learning Frameworks," https://www.edureka.co/blog/keras-vs-tensorflow-vs-pytorch/, Accessed: 2019-05-23.

Arquitecturas, diseños de referencia y plataformas

Mejora de un Código de Corrección de Errores para tolerar fallos adyacentes bidimensionales

J. Gracia-Morán*, L.J. Saiz-Adalid*, D. Gil-Tomás*, J.C. Baraza-Calvo*, P.J. Gil-Vicente*

***Resumen*—Durante estos últimos años, el desarrollo tecnológico ha permitido aumentar la escala de integración de los circuitos integrados. En particular, este aumento ha posibilitado la creación de sistemas de memoria de gran capacidad. Sin embargo, también ha provocado un incremento en su tasa de fallos, aumentando la probabilidad de que se produzcan *Single Cell Upsets* (SCUs) o *Multiple Cell Upsets* (MCUs).**

Una posible solución para tolerar estos errores es el uso de Códigos de Corrección de Errores (del inglés *Error Correction Codes* – ECCs). Dependiendo del ECC introducido, es posible corregir una gran variedad de tipos de errores, teniendo en cuenta que la introducción de un ECC implica una serie de sobrecargas a considerar, sobre todo cuando el ECC se utiliza en aplicaciones empotradas.

En un trabajo anterior presentamos un ECC diseñado para corregir fallos adyacentes, apto para aplicaciones empotradas. En este trabajo se presenta una mejora de este ECC que amplía la cobertura de error frente a fallos adyacentes sin aumentar el número de bits extra necesarios para corregirlos.

***Palabras Clave*—Códigos de Corrección de Errores, Multiple Cell Upsets, Tolerancia a Fallos, Confiabilidad**

I. Introducción

La continua reducción de tamaño de la tecnología CMOS, que ha permitido obtener sistemas de memoria con una gran capacidad de almacenamiento, también ha provocado un aumento en la tasa de errores en este elemento [1][2]. En este sentido, el impacto de una partícula de radiación cósmica puede provocar el cambio en una única celda de memoria (evento conocido como *Single Cell Upset* o SCU) o en varias celdas de memoria (*Multiple Cell Upset* o MCU) [3][4][5][6][7].

Una posible solución para la protección de sistemas de memoria son los Códigos de Corrección de Errores (del inglés *Error Correction Codes* – ECCs), como pueden ser, por ejemplo, los códigos SEC o SEC-DED [8][9][10]. Los códigos SEC (*Single Error Correction*) pueden corregir un error en una única celda de memoria, mientras que los códigos SEC-DED (*Single Error Correction – Double Error Detection*) pueden corregir un error en una celda de memoria, así como detectar dos errores en dos celdas independientes.

A medida que el número de errores crece, también debe aumentar la cobertura de error de los ECCs añadidos. Por ejemplo, en aplicaciones críticas, se utilizan códigos más complejos y sofisticados [11][12][13][14][15][16][17][18]. De todos ellos, nosotros nos hemos centrado en los códigos matriciales [14][15] por su facilidad de implementación, que provoca una baja sobrecarga, y la posibilidad de poder adaptar dichos códigos a diferentes tipos de error.

Originalmente, los códigos matriciales [14][15] combinan códigos de Hamming [9] con una verificación de paridad en un formato bidimensional, lo que permite la corrección y/o detección de dos bits erróneos.

Sin embargo, un hecho a considerar, sobre todo en aplicaciones empotradas, es que la incorporación de un mecanismo de tolerancia a fallos a un sistema conlleva añadir una determinada sobrecarga al mismo. En concreto, cuando se incluye un ECC en un sistema de memoria, hay que tener en cuenta, en primer lugar, el número de bits adicionales que se usan para detectar y/o corregir los posibles errores ocurridos, y que se añaden a cada palabra de datos almacenada en la memoria. En este sentido, el número de bits redundantes debe ser lo más bajo posible. Por ejemplo, si se emplea un ECC con un 100% de redundancia en una memoria de 2GB, solo 1GB estará disponible para almacenar la carga (los datos "limpios"), mientras que el 1GB restante es requerido para los bits de código.

En segundo lugar, también hay que tener en cuenta la sobrecarga introducida con respecto al área de silicio ocupada, potencia consumida y retardo de los circuitos de codificación y decodificación. Esta sobrecarga es dependiente de la complejidad de dichos circuitos.

En un trabajo anterior [19], presentamos un nuevo ECC matricial, que denominamos FUEC-M. Este ECC presenta una baja redundancia, mejorando la cobertura de errores con respecto a los códigos matriciales originales. FUEC-M está diseñado para corregir una serie de patrones de errores adyacentes.

En este trabajo presentamos la mejora de FUEC-M (que hemos denominado FUEC-ME). Este nuevo ECC matricial permite corregir más patrones de fallos adyacentes que el código FUEC-M, pero manteniendo la misma redundancia.

El presente trabajo lo hemos organizado de la siguiente manera. La Sección II resume el funcionamiento de los códigos matriciales, así como del código FUEC-M. La Sección III introduce el nuevo código FUEC-ME. La Sección IV describe los diferentes resultados obtenidos durante la evaluación de los ECCs introducidos en las Secciones II y III. Y finalmente, la Sección V concluye este trabajo.

II. Códigos de Corrección de Errores Matriciales

A. Introducción a los códigos matriciales

Tradicionalmente, se define un ECC matricial como aquel código corrector de errores que organiza los datos de forma bidimensional, de tal manera que la combinación de dos, o más, ECCs permite aumentar las capacidades de detección y/o corrección de distintos tipos de errores. En el caso concreto de errores adyacentes, se deben tener en cuenta tanto la dimensión horizontal como la dimensión vertical [14][15][16][25][26][27].

* Instituto ITACA, Universitat Politècnica de València
e-mail: { jgracia, ljsaiz, dgil, jcbaraza, pgil }@itaca.upv.es

Un ejemplo típico es el presentado en la Fig. 1 (extraído de [15]), donde X_i representa los bits de datos, C_j son los bits de control horizontales (calculados mediante un código de Hamming), y P_k son los bits de paridad vertical (calculados mediante paridad par).

X_1	X_2	X_3	X_4	C_1	C_2	C_3
X_5	X_6	X_7	X_8	C_4	C_5	C_6
X_9	X_{10}	X_{11}	X_{12}	C_7	C_8	C_9
X_{13}	X_{14}	X_{15}	X_{16}	C_{10}	C_{11}	C_{12}
P_1	P_2	P_3	P_4			

Fig. 1. Esquema de código matricial [15].

El funcionamiento básico de este código matricial es el siguiente. Los bits de datos (X_i) se dividen en grupos de 4 bits. Cada grupo está codificado por un código de Hamming [9] que utiliza 3 bits redundantes para proteger 4 bits de datos, y que sirve para generar los diferentes C_j. Por último, un conjunto de bits de paridad par (P_k) completa la matriz. La Tabla I muestra las fórmulas (extraídas de [15]) para calcular los bits de código (Tabla I-a) y los bits de paridad (Tabla I-b). De esta manera, un error simple se corrige mediante los bits C_j (código de Hamming), mientras que para corregir errores adyacentes, se utilizan los bits C_j junto con los bits P_k.

TABLA I
FÓRMULAS PARA EL CÁLCULO DE LOS BITS DE CÓDIGO Y DE PARIDAD [15]

$$C_1 = X_2 \oplus X_3 \oplus X_4$$
$$C_2 = X_1 \oplus X_3 \oplus X_4$$
$$C_3 = X_1 \oplus X_2 \oplus X_4$$
$$C_4 = X_6 \oplus X_7 \oplus X_8$$
$$C_5 = X_5 \oplus X_7 \oplus X_8$$
$$C_6 = X_5 \oplus X_6 \oplus X_8$$
$$C_7 = X_{10} \oplus X_{11} \oplus X_{12}$$
$$C_8 = X_9 \oplus X_{11} \oplus X_{12}$$
$$C_9 = X_9 \oplus X_{10} \oplus X_{12}$$
$$C_{10} = X_{14} \oplus X_{15} \oplus X_{16}$$
$$C_{11} = X_{13} \oplus X_{15} \oplus X_{16}$$
$$C_{12} = X_{13} \oplus X_{14} \oplus X_{16}$$

a)

$$p_1 = X_1 \oplus X_5 \oplus X_9 \oplus X_{13}$$
$$p_2 = X_2 \oplus X_6 \oplus X_{10} \oplus X_{14}$$
$$p_3 = X_3 \oplus X_7 \oplus X_{11} \oplus X_{15}$$
$$p_4 = X_4 \oplus X_8 \oplus X_{12} \oplus X_{16}$$

b)

Este ECC presenta dos grandes ventajas. Por un lado, mejora las prestaciones del código de Hamming y de las verificaciones de paridad. La combinación de ambos códigos permite aumentar su tasa de detección y corrección, ya que este código matricial es capaz de corregir todos los errores simples, así como de corregir o detectar todos los errores adyacentes de 2 bits.

Por otro lado, la sobrecarga provocada al utilizar códigos de Hamming y bits de paridad no es muy elevada, pues los circuitos para codificar y decodificar estos dos ECCs suelen ser muy eficientes.

Sin embargo, este ECC introduce una redundancia muy elevada, lo que provoca un aumento en la memoria requerida para los bits de código. Si calculamos la redundancia mediante (1), podemos ver que este código presenta un 100% de redundancia. Es decir, en una memoria, la mitad de los bits servirán para almacenar datos, y la otra mitad para almacenar los bits de código.

$$Redundancia = \frac{No.\ bits\ código}{No.\ bits\ datos} \times 100 \quad (1)$$

B. *Código FUEC-M*

En un trabajo anterior presentamos FUEC-M [19], un código bidimensional que mejoraba las prestaciones del código matricial [15]. Con el esquema que se muestra en la Fig. 2 (donde X_i representan los bits de datos y C_j los bits de control calculados mediante el código FUEC-M), este nuevo código matricial introduce un número menor de bits de redundancia, tal y como se puede ver en Tabla II. En esta tabla (extraída de [19]) se pueden ver las fórmulas que se aplican para obtener los bits de código.

En concreto, si aplicamos la fórmula (1), la redundancia de este ECC es de 56'25%. Esta baja redundancia implica que existirá una mayor disponibilidad en la memoria para almacenar bits de datos. Por ejemplo, en el caso del código matricial [15], si se tiene un chip de memoria de 1GB, solo 512MB estarán disponibles para almacenar bits de datos, pues los 512MB restantes son necesarios para almacenar los bits de código. En el caso del código FUEC-M, sólo 370MB son necesarios para almacenar los bits de código, pudiéndose dedicar sobre 655MB al almacenamiento de los bits de datos.

C_0	C_1	C_2	C_3	C_4
C_5	C_6	C_7	C_8	X_0
X_1	X_2	X_3	X_4	X_5
X_6	X_7	X_8	X_9	X_{10}
X_{11}	X_{12}	X_{13}	X_{14}	X_{15}

Fig. 2. Esquema de nuestro código matricial [19].

TABLA II
FÓRMULAS PARA EL CÁLCULO DE LOS BITS DE CÓDIGO [19]

$$C_0 = X_0 \oplus X_1 \oplus X_7 \oplus X_{10}$$
$$C_1 = X_2 \oplus X_3 \oplus X_4 \oplus X_8$$
$$C_2 = X_0 \oplus X_5 \oplus X_6 \oplus X_{12}$$
$$C_3 = X_1 \oplus X_4 \oplus X_{11} \oplus X_{14}$$
$$C_4 = X_2 \oplus X_5 \oplus X_9 \oplus X_{15}$$
$$C_5 = X_3 \oplus X_6$$
$$C_6 = X_7 \oplus X_9 \oplus X_{11} \oplus X_{13}$$
$$C_7 = X_8 \oplus X_{10} \oplus X_{14}$$
$$C_8 = X_{12} \oplus X_{13} \oplus X_{15}$$

Otra diferencia entre el código matricial [15] y FUEC-M es su cobertura de errores. En concreto, FUEC-M es capaz de corregir errores simples, errores dobles adyacentes tanto en horizontal como en vertical, y errores cuádruples adyacentes en cuadrados de 2x2.

Más información sobre el diseño de Códigos de Corrección de Errores en general, y sobre FUEC-M en particular, puede verse en [19].

III. CÓDIGO DE CORRECCIÓN DE ERRORES FUEC-ME

El nuevo código corrector de errores presentado en este trabajo (denominado FUEC-ME) amplía las características de corrección de errores de FUEC-M utilizando el mismo número de bits redundantes. En concreto, este nuevo código es capaz de corregir errores simples, errores dobles adyacentes tanto en horizontal como en vertical, y errores cuádruples adyacentes en cuadrados de 2x2 tal y como hace FUEC-M. Y, además, también puede corregir errores adyacentes de 3, 4 y 5 bits, tanto en horizontal como en vertical, y errores cuadrados de 3x2, 2x3 y 3x3 bits.

A la hora de diseñar un ECC, se deben fijar una serie de parámetros. En concreto, el conjunto de errores a tolerar, el

número de bits de datos y el número de bits de código. Con estos tres parámetros, nuestra herramienta puede generar la matriz de paridad **H** que define a un ECC [8][19][20]. En concreto, la matriz de paridad **H** que define al código FUEC-ME se puede ver en la Fig. 3.

$$H = \begin{array}{c} C_0 C_1 \cdots\cdots C_8 X_0 \cdots\cdots\cdots\cdots X_{15} \\ \left|\begin{array}{l} 100000000110001001000000 \\ 010000000011001000001000 \\ 001000001000100000010001 \\ 000100000100000100100100 \\ 000010000001010000000011 \\ 000001000000100000101001 \\ 000000100000001000100101 \\ 000000010000000110110000 \\ 000000001000000010011010 \end{array}\right| \end{array}$$

Fig. 3. Matriz de paridad **H** del código FUEC-ME.

A partir de **H** se pueden obtener fácilmente las fórmulas para la codificación y para la obtención del síndrome, tal y como se puede ver en la Tabla III, en donde la Tabla III-a muestra las fórmulas para la obtención del código que se almacena con cada una de las palabras de datos, mientras que la Tabla III-b muestra las fórmulas para la obtención del síndrome que se utilizará para la corrección de errores. En concreto, un '1' en la matriz de paridad **H** indica que ese bit en particular se debe tener en cuenta a la hora de calcular el bit de código. Por ejemplo, para calcular el bit S_0 hay que incluir en la fórmula correspondiente los bits C_0, X_0, X_1, X_5 y X_8. Por otro lado, para calcular el bit C_0, los bits a incluir son X_0, X_1, X_5 y X_8. Se puede ver más información en [8][19][20][28].

Si comparamos la Tabla II con la Tabla III, vemos que las fórmulas obtenidas para el código FUEC-ME son más complejas. Este es un resultado previsible, pues hemos aumentado la cobertura de errores sin aumentar el número de bits redundantes. Este hecho también hace prever que la sobrecarga, en cuanto al área de silicio ocupada, potencia consumida y retardo, será mayor para el código FUEC-ME.

TABLA III
FÓRMULAS OBTENIDAS A PARTIR DE LA MATRIZ DE PARIDAD DE LA FIG. 3

$$\begin{aligned} C_0 &= X_0 \oplus X_1 \oplus X_5 \oplus X_8 \\ C_1 &= X_2 \oplus X_3 \oplus X_6 \oplus X_{12} \\ C_2 &= X_0 \oplus X_4 \oplus X_{11} \oplus X_{15} \\ C_3 &= X_1 \oplus X_7 \oplus X_{10} \oplus X_{13} \\ C_4 &= X_2 \oplus X_4 \oplus X_{14} \oplus X_{15} \\ C_5 &= X_3 \oplus X_9 \oplus X_{11} \oplus X_{15} \\ C_6 &= X_5 \oplus X_{10} \oplus X_{13} \oplus X_{15} \\ C_7 &= X_6 \oplus X_7 \oplus X_9 \oplus X_{10} \\ C_8 &= X_8 \oplus X_{11} \oplus X_{12} \oplus X_{14} \end{aligned}$$

a)

$$\begin{aligned} s_0 &= C_0 \oplus X_0 \oplus X_1 \oplus X_5 \oplus X_8 \\ s_1 &= C_1 \oplus X_2 \oplus X_3 \oplus X_6 \oplus X_{12} \\ s_2 &= C_2 \oplus X_0 \oplus X_4 \oplus X_{11} \oplus X_{15} \\ s_3 &= C_3 \oplus X_1 \oplus X_7 \oplus X_{10} \oplus X_{13} \\ s_4 &= C_4 \oplus X_2 \oplus X_4 \oplus X_{14} \oplus X_{15} \\ s_5 &= C_5 \oplus X_3 \oplus X_9 \oplus X_{11} \oplus X_{15} \\ s_6 &= C_6 \oplus X_5 \oplus X_{10} \oplus X_{13} \oplus X_{15} \\ s_7 &= C_7 \oplus X_6 \oplus X_7 \oplus X_9 \oplus X_{10} \\ s_8 &= C_8 \oplus X_8 \oplus X_{11} \oplus X_{12} \oplus X_{14} \end{aligned}$$

b)

IV. EVALUACIÓN DE LOS ECCS

En esta sección comenzaremos introduciendo los modelos de error utilizados habitualmente en teoría de códigos, para después presentar los resultados obtenidos durante la evaluación de los diferentes ECCs presentados anteriormente: el código matricial [15], FUEC-M [19] y FUEC-ME.

Esta evaluación se ha realizado mediante dos procesos diferentes. En primer lugar, se han inyectado diferentes tipos de errores en los modelos en C de los ECCs bajo estudio. Con este método, es posible evaluar su cobertura de errores. En un segundo paso, los diferentes circuitos codificadores y decodificadores se han implementado en VHDL, pudiendo así sintetizarlos con el fin de estimar la sobrecarga introducida con respecto al área de silicio, potencia consumida y retardo.

A. Modelos de Error

Se define, en teoría de códigos, el término *error aleatorio* (del inglés *random error*) como uno o más bits de una palabra codificada erróneo, siendo la palabra codificada la formada por los bits de datos más los bits de código generados por el ECC.

Los *errores aleatorios* pueden ser *simples* o *múltiples*. Los *errores simples* afectan a una única celda de memoria. Se producen comúnmente por lo que en inglés se conoce como *Single Event Upsets* (SEU, en memorias RAM) o *Single Event Transients* (SET, en lógica combinacional) [22]. Por otra parte, los *errores múltiples* afectan a más de una celda de memoria, siendo cada vez más frecuentes [3][4][5][6][7]. Se pueden generar *errores adyacentes*, es decir, errores múltiples donde todos los bits erróneos son contiguos, cuando una partícula cósmica impacta en una celda de memoria [23]. Este es el tipo de error múltiple más frecuente [4][24].

Si tenemos en cuenta el nivel de abstracción que estamos utilizando, en nuestro caso el nivel lógico, el diseño de los experimentos de inyección se puede simplificar. En concreto, la idea básica es cambiar el estado del bit afectado por el error, invirtiendo su valor. En la siguiente sección se explica con más detalle el funcionamiento de la herramienta de inyección de fallos y el procedimiento utilizado para comprobar los resultados.

B. Evaluación de la cobertura de errores

Mediante la utilización de un inyector de fallos basado en simulación, el cual hemos desarrollado en trabajos anteriores [28], hemos podido estudiar la cobertura de errores de los ECCs presentados en secciones anteriores. El esquema básico de la herramienta de inyección se muestra en la Fig. 4.

Con esta herramienta hemos podido inyectar diferentes tipos de errores, pudiendo comprobar si la palabra de datos final es correcta o no. Además, la herramienta también puede generar la señal NRE (*Error No Recuperable*, del inglés *Non Recoverable Error*) cuando se detecta un error que no se puede corregir. De esta forma, e inyectando todos los errores de un tamaño y modelo dado, es posible contar el número de errores corregidos y/o detectados con respecto al número total de errores posibles. Así, se puede calcular la cobertura de error de cada código.

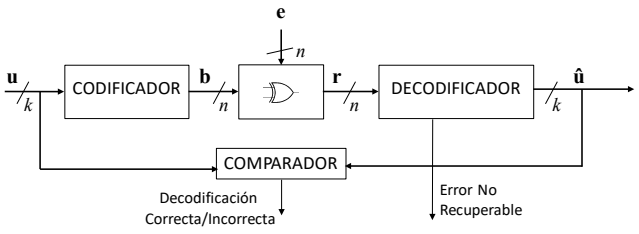


Fig. 4. Diagrama de bloques de la herramienta de inyección de fallos [28].

Un detalle a tener en cuenta es que no hemos inyectado errores según su probabilidad de ocurrencia, sino que se ha inyectado cada tipo de error (errores simples o errores adyacentes de diferentes longitudes) en todos los bits de la

palabra de código. De esta forma, se han podido verificar las capacidades de corrección de errores de los diferentes códigos.

Los diferentes bloques de la herramienta de inyección de la Fig. 4 se han implementado en C. Se han utilizado operadores de lógica de bit para poder simular de forma precisa el comportamiento del hardware. A partir de la matriz de paridad **H** (como, por ejemplo, la mostrada en la Fig. 3) se obtienen fácilmente los circuitos codificadores y decodificadores de los diferentes ECCs. Estos circuitos se implementan en C mediante funciones, por lo que cambiar de un ECC a otro es tan sencillo como ajustar las longitudes de palabra y reemplazar las funciones de codificación y decodificación para el nuevo ECC.

Con respecto al tipo de error inyectado, en este trabajo se han inyectado los mismos errores que en [19], con el fin de poder comparar el comportamiento del código FUEC-ME con respecto al código FUEC-M y el código matricial [15].

Con los datos obtenidos en los diferentes experimentos de inyección, y utilizando la fórmula (2), hemos calculado la cobertura de corrección de errores de los diferentes ECCs:

$$C_{correc} = \frac{Errores\_Corregidos}{Errores\_Inyectados} \times 100 \qquad (2)$$

donde *Errores_Corregidos* es el número de errores corregidos por cada ECC, mientras que *Errores_Inyectados* es el número de errores inyectados de un determinado patrón.

Los resultados obtenidos se pueden ver en la Tabla IV. Como era de esperar, el nuevo código FUEC-ME es capaz de corregir todos los errores que hemos inyectado: errores simples, errores adyacentes de longitudes de 2 a 5 bits, tanto en horizontal como en vertical, y diferentes patrones cuadrados de errores adyacentes, como son 2x2, 2x3, 3x2 y 3x3. Como se puede ver, estos resultados mejoran los obtenidos por el código FUEC-M [19], tal y como habíamos planteado al diseñar el código FUEC-ME.

En resumen, el código FUEC-ME permite la corrección de diferentes tipos de errores adyacentes con una redundancia muy baja. Si el comportamiento de la memoria a proteger es afectado por este tipo de errores (errores adyacentes de más de dos bits con diferentes patrones), el código FUEC-ME es una buena alternativa, por dos razones principales: i) su baja redundancia; y ii) su alta cobertura de corrección de errores.

C. Resultados de la síntesis de los ECCs

Hemos visto en la Sección III que el código FUEC-ME presenta la misma redundancia que el código FUEC-M, y en la Sección IV.B hemos comprobado que el código FUEC-ME puede corregir un mayor número y tipo de errores adyacentes que el código FUEC-M.

En este apartado vamos a estudiar la sobrecarga, con respecto al área de silicio, potencia consumida y retardo, que introduce el código FUEC-ME, comparándola con la introducida por el código FUEC-M y el código matricial [15]. Para ello, se han sintetizado los diferentes circuitos de codificación y decodificación de los ECCs estudiados. En concreto, en primer lugar estos circuitos se han implementado en VHDL, para a continuación, y utilizando el software CADENCE [29], se ha realizado una síntesis lógica para la tecnología de 45 nm, mediante el uso de la biblioteca NanGate FreePDK45 [30][31].

TABLA IV
PORCENTAJES DE ERRORES CORREGIDOS

	Código matricial [15]	**Código FUEC-M [19]**	**Código FUEC-ME**
Patrón de Errores Horizontal			
Longitud del Error	% Errores Corregidos	% Errores Corregidos	% Errores Corregidos
1	100,00	100,00	100,00
2	89,15	100,00	100,00
3	45,45	0,00	100,00
4	5,88	0,00	100,00
5	0,00	0,00	100,00
Patrón de Errores Vertical			
Longitud del Error	% Errores Corregidos	% Errores Corregidos	% Errores Corregidos
1	100,00	100,00	100,00
2	36,00	100,00	100,00
3	100,00	6,67	100,00
4	27,27	0,00	100,00
5	100,00	0,00	100,00
Patrón de Errores Cuadrado			
Longitud del Error	% Errores Corregidos	% Errores Corregidos	% Errores Corregidos
2x2	28,57	100,00	100,00
3x2	26,67	0,00	100,00
2x3	17,65	0,00	100,00
3x3	16,67	0,00	100,00

La Fig. 5 muestra el área de silicio ocupada por los diferentes circuitos. Como se puede ver, el codificador del código FUEC-ME introduce una sobrecarga similar al del código FUEC-M, y menor que la del código matricial [15]. En cuanto al decodificador, la mayor sobrecarga la presenta el código FUEC-ME. Este es un resultado esperado pues este código también presenta una mayor cobertura de errores, tal y como se ha visto en la Tabla IV.

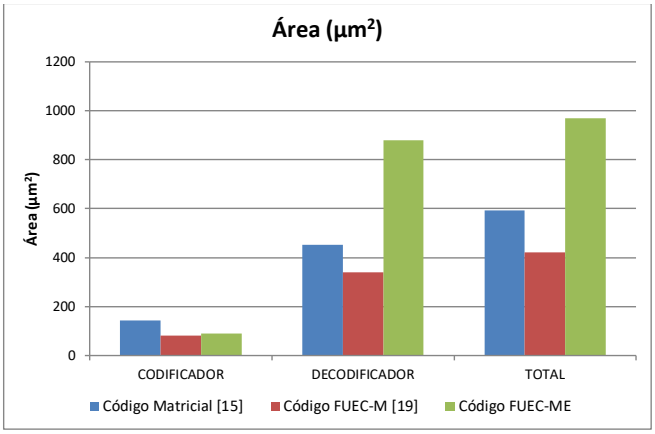


Fig. 5. Área ocupada por los diferentes ECCs.

La Fig. 6 por su parte muestra la potencia consumida por los diferentes ECCs. El consumo está directamente relacionado con el área ocupada. De esta forma, podemos comprobar que el codificador del código FUEC-ME presenta un consumo ligeramente superior al consumo del código FUEC-M, y ligeramente inferior al del código matricial [15]. En cuanto al decodificador, al presentar un área mayor, también presenta un mayor consumo de potencia.

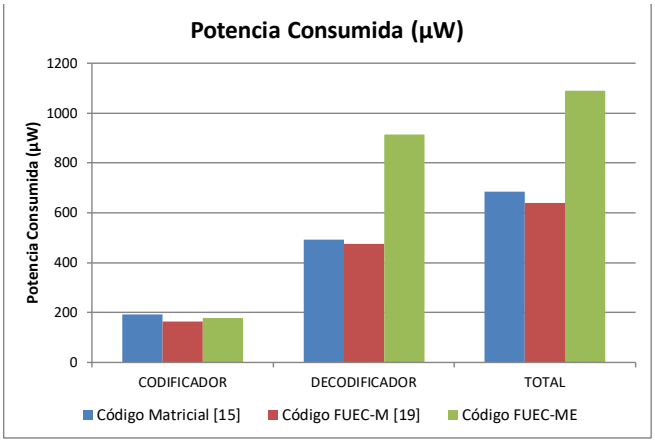


Fig. 6. Potencia consumida por los diferentes ECCs.

Finalmente, la Fig. 7 muestra la sobrecarga temporal de los tres códigos. Al igual que con el área y la potencia consumida, la mayor cobertura de errores del código FUEC-ME provoca que el retardo introducido sea mayor. Este hecho se puede ver claramente en el caso del decodificador, ya que el número de operaciones a realizar para obtener la cobertura de error tan alta que presenta el código FUEC-ME provoca un mayor retardo.

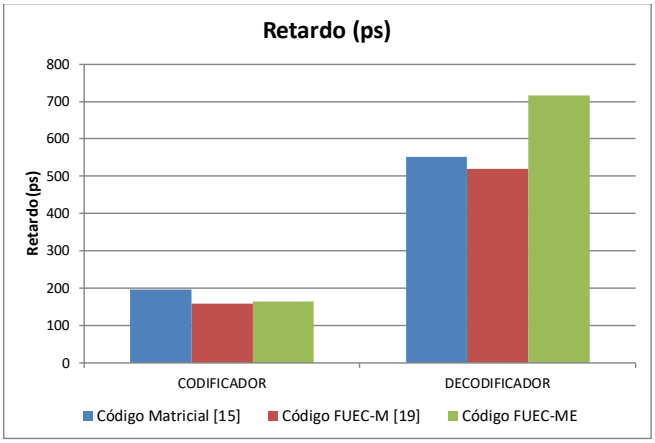


Fig. 7. Sobrecarga temporal de los diferentes ECCs.

Como se ha visto, el código FUEC-ME presenta una mayor sobrecarga que el código FUEC-M y el código matricial [15], principalmente en el decodificador. Esta mayor sobrecarga se debe, por un lado, al bajo número de bits redundantes del código FUEC-ME, y, por otra parte, a la gran capacidad de corrección de errores de este código.

En este sentido, y teniendo en cuenta que el número de bits de código es el mismo para los dos ECCs, podemos concluir que el código FUEC-M será adecuado para aquellas aplicaciones en las que el número de errores adyacentes previstos sea como máximo 2. Si este número es mayor, sería mejor utilizar el código FUEC-ME. En cualquier caso, sí que es desaconsejable el uso del código matricial [15], pues presenta una cobertura de errores menor y una mayor redundancia.

V. Conclusiones

En este trabajo hemos mejorado un código corrector de errores de tipo matricial. Esta mejora introduce una baja redundancia junto con una elevada corrección de errores, que permite tolerar errores simples y diferentes patrones de errores múltiples adyacentes.

Hemos comprobado la eficiencia de esta mejora (el código FUEC-ME) comparando tanto la cobertura de corrección de errores como la sobrecarga introducida en función del área de silicio ocupada, potencia consumida y retardo con respecto al código original (el código FUEC-M) y otro código matricial bien conocido.

Con respecto a la cobertura de corrección de errores, el código FUEC-ME supera ampliamente las coberturas de corrección de errores de los ECCs comparados. En concreto, el código FUEC-ME es capaz de corregir el 100% de los errores simples y de errores adyacentes de longitud de 2 a 5 bits, tanto en horizontal como en vertical, y diferentes patrones cuadrados de errores adyacentes, como son 2x2, 2x3, 3x2 y 3x3.

Por otro lado, el código FUEC-ME introduce una mayor sobrecarga con respecto al área ocupada, potencia consumida y retardo de los circuitos de codificación y decodificación que los códigos comparados, debida principalmente a su mayor capacidad de corrección de errores.

En general, el código FUEC-ME complementa al código FUEC-M, siendo este último una opción adecuada para aplicaciones en las que se esperen errores adyacentes dobles, mientras que el código FUEC-ME se puede emplear en aquellas aplicaciones en las que se producen errores adyacentes de mayor tamaño.

En trabajos futuros queremos seguir desarrollando ECCs con una baja redundancia o que disminuyan la sobrecarga en el área de silicio ocupada, la potencia consumida y el retardo, manteniendo, o incluso mejorando, la cobertura de error. Por otro lado, también se quiere desarrollar otros códigos centrados en los errores múltiples adyacentes de mayor tamaño, que se espera que tengan un impacto cada vez más importante. También queremos estudiar los cambios necesarios en la herramienta de generación de ECCs que permita diseñar códigos en función de la cobertura de errores, el retardo deseado y el área máxima que ocuparán el codificador y el decodificador.

Agradecimientos

El presente trabajo ha sido parcialmente financiado por el gobierno de España mediante el proyecto de investigación TIN2016-81075-R, y por el Vicerrectorado de Investigación, Innovación y Transferencia de la Universitat Politècnica de València (UPV), dentro del programa Primeros Proyectos de Investigación (PAID-06-18) bajo la referencia 200190032.

Referencias

[1] The International Technology Roadmap for Semiconductors 2015. [Online]. Available at: http://www.itrs2.net/itrs-reports.html.

[2] S.K. Kurinec and K. Iniewsky. *Nanoscale Semiconductor Memories: Technology and Application*, CRC Press, Taylor & Francis Group, 2014.

[3] E. Ibe, H. Taniguchi, Y. Yahagi, K. Shimbo, and T. Toba, "Impact of scaling on neutron-induced soft error in SRAMs from a 250 nm to a 22 nm design rule", IEEE Trans. Electron Devices, vol. 57, no. 7, pp. 1527–1538, July 2010.

[4] G. Tsiligiannis et. al., "Multiple Cell Upset Classification in Commercial SRAMs", IEEE Transactions on Nuclear Science, vol. 61, no. 4, August 2014.

[5] G.I. Zebrev, "Multiple Cell Upset Cross-Section Uncertainty in Nanoscale Memories: Microdosimetric Approach", 15th European

Conference on Radiation and its Effects on Components and Systems (RADECS 2015), September 2015.

[6] N.G. Chechenin and M. Sajid, "Multiple cell upsets rate estimation for 65 nm SRAM bit-cell in space radiation environment", 3rd International Conference and Exhibition on Satellite & Space Missions, May 2017.

[7] N.N. Mahatme, B.L. Bhuva, Y.P. Fang, and A.S. Oates, "Impact of strained-Si PMOS transistors on SRAM soft error rates", IEEE Trans. on Nuclear Science, vol. 59, no. 4, pp. 845–850, August 2012.

[8] E. Fujiwara, Code Design for Dependable Systems: Theory and Practical Application, Ed. Wiley-Interscience, 2006.

[9] R. W. Hamming, "Error detecting and error correcting codes," Bell System Technical Journal, vol. 29, pp. 147–160, 1950.

[10] C.L. Chen and M.Y. Hsiao, "Error-correcting codes for semiconductor memory applications: a state-of-the-art review", IBM Journal of Research and Development, vol. 58, no. 2, pp. 124–134, March 1984.

[11] G.C. Cardarilli, M. Ottavi, S. Pontarelli, M. Re, and A. Salsano, "Fault Tolerant Solid State Mass Memory for Space Applications", IEEE Trans. on Aerospace and Electronic Systems, vol. 41, no. 4, pp. 1353–1372, October 2005.

[12] S. Pontarelli, G.C. Cardarilli, M. Re and A. Salsano, "Error correction codes for SEU and SEFI tolerant memory systems", 24th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT 2009), pp. 425-430, 2009.

[13] A. Sánchez-Macián, P. Reviriego, J. Tabero, A. Regadío, and J.A. Maestro, "SEFI protection for Nanosat 16-bit Chip On-Board Computer Memories", IEEE Transactions on Device and Materials Reliability, DOI 10.1109/TDMR.2017.2750718, 2017.

[14] C. Argyrides, D.K. Pradhan, and T. Kocak, "Matrix codes for reliable and cost efficient memory chips", IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 19, nº 3, pp.420–428, March 2011.

[15] C. Argyrides, H.R. Zarandi and D.K. Pradhan, "Matrix Codes: Multiple Bit Upsets Tolerant Method for SRAM Memories", 22nd IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, 2007.

[16] H.S. de Castro, et al. "A correction code for multiple cells upsets in memory devices for space applications", 2016 14th IEEE International New Circuits and Systems Conference (NEWCAS 2016), pp.1–4, June 2016.

[17] S. Ahmad, M. Zahra. S.Z. Farooq, and A. Zafar, "Comparison of EDAC schemes for DDR memory in space applications", 2013 International Conference on Aerospace Science & Engineering (ICASE 2013), August 2013.

[18] D.E. Muller, "Application of boolean algebra to switching circuit design and to error detection", IRE Transactions on Electronic Computers, vol. 3, pp. 6–12, 1954.

[19] J. Gracia-Morán, L.J. Saiz-Adalid, D. Gil-Tomás, P.J. Gil-Vicente, "Un nuevo Código de Corrección de Errores matricial con baja redundancia", III Jornadas de Computación Empotrada y Reconfigurable (JCER2018), Jornadas SARTECO, pp. 561-566, Septiembre 2018.

[20] L.J. Saiz-Adalid et al., "Flexible Unequal Error Control codes with selectable error detection and correction levels," 32th International Conference on Computer Safety, Reliability and Security (SAFECOMP), pp. 178–189, Septiembre 2013.

[21] A. Neubauer, J. Freudenberger, and V. Kühn, Coding Theory: Algorithms, Architectures and Applications. John Wiley & Sons, 2007.

[22] K.A. LaBel, "Proton single event effects (SEE) guideline" submitted for publication on the NASA Electronic Parts and Packaging (NEPP) Program web site, August 2009. Available online at https://nepp.nasa.gov/files/18365/Proton_RHAGuide_NASAAug09.pdf

[23] M. Murat, A. Akkerman, and J. Barak, "Electron and ion tracks in silicon: Spatial and temporal evolution," IEEE Transactions on Nuclear Science, vol. 55, no. 6, pp. 3046–3054, December 2008.

[24] M. Wirthlin, D. Lee, G. Swift, and H. Quinn, "A method and case study on identifying physically adjacent multiple-cell upsets using 28-nm, interleaved and SECDED-protected arrays," IEEE Transactions on Nuclear Science, vol. 61, no. 6, pp. 3080–3087, Dec. 2014.

[25] S. Liu, L. Xiao, J. Li, Y. Zhou, and Z. Mao, "Low Redundancy Matrix-Based codes for Adjacent Error Correction with Parity Sharing", 2017 18th International Symposium on Quality Electronic Design (ISQED 2017, March 2017.

[26] P. Reviriego and J.A. Maestro, "Efficient Error Detection Codes for Multiple-Bit Upset Correction in SRAMs with BICS", ACM Transactions on Design Automation of Electronic Systems (TODAES) Vol. 14 nº 1, January 2009.

[27] M. Zhu, L. Xiao, S. Li, and Y. Zhang, "Efficient Two-Dimensional Error Codes for Multiple Bit Upsets Mitigation in Memory", 2010 25th International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT 2010), pp. 129-135, October 2010.

[28] J. Gracia-Morán , L.J. Saiz-Adalid, D. Gil-Tomás, P.J. Gil-Vicente, "Improving Error Correction Codes for Multiple-Cell Upsets in Space Applications", IEEE Transactions on VLSI Systems, vol. 26(10), pp. 2132-2142, October 2018.

[29] *Cadence: EDA Tools and IP for System Design Enablement*. [Online]. Disponible en: https://www.cadence.com/

[30] J.E Stine et al., "FreePDK: An Open-Source Variation-Aware Design Kit", IEEE International Conference on Microelectronic Systems Education (MSE'07), June 2007.

[31] *NanGate FreePDK45 Open Cell Library*. Disponible en: https://www.eda.ncsu.edu/wiki/FreePDK45:Contents

Diseño de una Arquitectura de Flujo de Datos para la Estimación de Movimiento en Tiempo Real Mediante la Técnica de Búsqueda Exhaustiva de Macrobloques

Eduardo Serrano, Jesús Barba, Julián Caba, M. Soledad Escolar,
Manuel J. Abaldea, Fernando Rincón, Juan Carlos López
Escuela Superior de Informática
Universidad de Castilla-La Mancha
Ciudad Real, España

Resumen— **La estimación de movimiento es la etapa de mayor coste computacional en los estándares de compresión de vídeo, los cuales tratan de reducir la cantidad de datos aprovechando la redundancia temporal existente entre dos *frames* consecutivos. Aunque el mecanismo es simple - dado un macrobloque el algoritmo tiene que encontrar su mejor emparejamiento dentro de una zona de búsqueda - su coste computacional es elevado. El mejor método, *Full Search* o búsqueda exhaustiva, utiliza un enfoque de fuerza bruta, el cual no es apropiado para aplicaciones en tiempo real.**

Este trabajo introduce una propuesta de arquitectura para FPGAs que implementa el algoritmo de estimación de movimiento mediante la técnica de búsqueda exhaustiva de macrobloques (*FSBM*). La solución propuesta ha sido modelada con la herramienta Vivado HLS en lenguaje C++, implementándose en la placa de prototipado ZC702 de Xilinx. El IP implementa una arquitectura de flujo de datos para el procesamiento en tiempo real de una fuente de vídeo. La arquitectura propuesta es configurable para adaptarse a diferentes alternativas.

Los resultados obtenidos en placa muestran una frecuencia de fotogramas de 746fps, 247fps y 110fps para resoluciones VGA, HD y Full HD, respectivamente. Con una frecuencia de reloj de 115Mhz, el IP consume en la FPGA un tercio de los FF y BRAMs y un 60 % de LUT.

Palabras clave— **FPGA, Síntesis de Alto Nivel, HLS, Estimación de Movimiento, Búsqueda Exhaustiva, Macro-bloques, Visión por Computador**

I. Introducción

MUCHOS estándares de codificación de vídeo (e.j. H.263, H.264, MPEG, ITU-T) hacen uso de técnicas de Estimación de Movimiento (ME) para eliminar redundancia temporal entre frames. Con el rápido crecimiento de las aplicaciones de vídeo y la mejora de resolución de las imágenes, la fase de ME se ha vuelto cada vez más crítica, alcanzando entre un 60 % y 80 % del tiempo total, dependiendo de la estrategia elegida para realizar la implememtación del algoritmo de Block Matching (BM).

Para cada macrobloque (MB), grupo de NxN píxeles, el algoritmo de BM trata de encontrar su mejor ajuste dentro de una zona de búsqueda ($(N+p)x(N+p)$), donde el parámetro p equivale al área de búsqueda. El objetivo es encontrar, en base a un criterio de similitud, la posición relativa del macrobloque dentro de la zona de búsqueda. Este proceso se repite para todos los macrobloques del frame.

El algoritmo *Full-Search Block Matching* (FSBM) consigue resultados de mayor precisión respecto a otras implementaciones, ya que realiza todas las comparaciones posibles de un macrobloque dentro de la zona de búsqueda. Pero esta precisión se traduce en un coste computacional prohibitivo para aplicaciones de tiempo real.

Sin embargo, las características que una plataforma basada en FPGA (*Field Programmable Gate Array*) ofrece permite solventar este inconveniente, alcanzando una ejecución en tiempo real.

Hasta hace poco tiempo, los lenguajes empleados en el desarrollo de soluciones para FPGA requerían de una alta curva de aprendizaje para introducirse en el diseño de tecnología de lógica reconfigurable. Sin embargo, con la aparición de entornos de trabajo HLS (del inglés *High Level Synthesis*), esta curva de aprendizaje se ha reducido, mejorando el acceso a esta tecnología.

En este trabajo, se detalla una arquitectura - diseñada en C++ con la herramienta Vivado HLS - que implementa el algoritmo FSBM, capaz de analizar vídeo en tiempo real.

Existen diferentes propuestas para solventar este problema, mayoritariamente se centran en la implementación de los módulos encargados de calcular la función de similitud entre dos macrobloques ([1], [2]). En cambio, el trabajo aquí presentado aborda una mejora global del algoritmo, introduciendo nuevas propuestas no contempladas anteriormente.

En [3] se aplica una técnica denominada *Online Arithmetic* que permite acelerar el cálculo del operador SAD. El acelerador propuesto es capaz de procesar 17.2 fps en formato VGA utilizando el dispositivo Virtex-II con una frecuencia de reloj de 425 Mhz y un tamaño de macrobloque de 16x16. Mientras que nuestra solución para VGA es capaz de procesar 186 fps con una frecuencia de reloj de 125 Mhz.

Las arquitecturas de tipo array sistólico son también consideradas como una solución óptima para la implementación del algoritmo FSBM por el uso óptimo de recursos, bajo consumo energético y configu-

rabilidad [4] [5]. Por ejemplo, [6] presenta un procesador escrito en VHDL capaz de procesar 60 fps (resolución CIF, reloj 192Mhz) en una FPGA Virtex-II para un tamaño de bloque de 8x8, ocupando un 11 % del área de la FPGA. Aunque es complicado proyectar una comparación, una configuración idéntica ($N = 16$ y $p = 8$) de la arquitectura propuesta es capaz de obtener el mismo rendimiento en resoluciones mucho más exigentes. Además, hay que tener en cuenta que en este trabajo se incluye toda la lógica que mueve la imagen desde memoria, y sería interesante ver cómo estas propuestas escalan al incorporar esta funcionalidad y aumenta la resolución del flujo de vídeo.

Nuno et. al validan en [7] varias estrategias de optimización para alcanzar cotas de eficiencia mayores en este tipo de arquitecturas sistólicas, todo ello sin bajar la calidad del resultado. Algunas de estas técnicas (ej. reducción de precisión de píxel) son de interés y su utilización en futuras versiones de la arquitectura con el objetivo de reducir recursos y mejorar los resultados del IP FSBM.

II. Descripción general de la arquitectura

El principal objetivo en el diseño del IP FSBM es realizar el cálculo de los vectores de movimiento, a partir de un flujo de datos, con el máximo rendimiento posible. Para este fin, se ha desarrollado una arquitectura de procesamiento de vídeo para la placa de prototipado ZC702 de Xilinx (ver Figura 1).

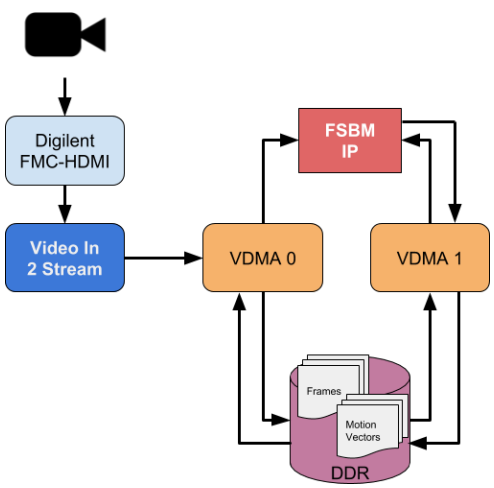


Fig. 1.Plataforma de procesamiento de vídeo para el cálculo de los vectores de movimiento.

Conforme se van recibiendo los frames desde la fuente de vídeo (tarjeta capturadora Digilent FMC-HDMI), éstos son almacenados en memoria DDR, en formato YUV 4:2:2 (16 bit/pixel). Dos Vídeo DMA se encargan de suministrar los datos de los dos frames que alimentan al IP FSBM. La sincronización entre los VDMA se realiza tanto a nivel hardware (configuración dinámica [1]) como software (configuración de un buffer circular).

Además de perseguir la máxima productividad (idealmente un ciclo de procesamiento por píxel), en el diseño se persigue que el uso de recursos sea el mínimo posible.

figure

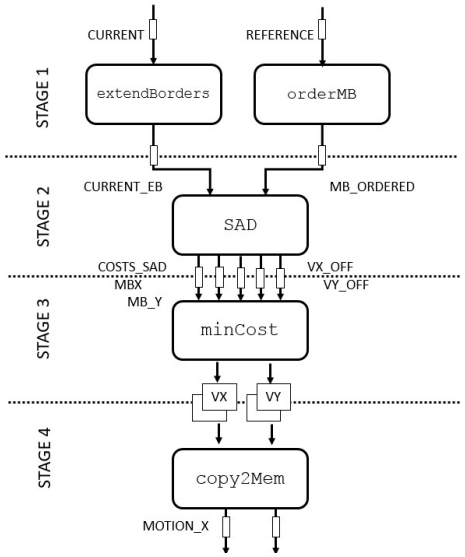


Fig. 2. Arquitectura propuesta de dataflow en el IP FSBM.

El modelo del IP FSBM consta de cuatro etapas, tal como se muestra en la Figura 2. A continuación se describe el trabajo realizado por cada etapa:

- Etapa 1: Recepción de datos y acomodación. Por un lado, se extienden los bordes del frame etiquetado como *CURRENT* para reducir la complejidad de los cálculos en etapas posteriores. Para el frame etiquetado como *REFERENCE* se realiza un reordenamiento y encapsulado de los macrobloques.
- Etapa 2: Función de coste. Para cada macrobloque de *REFERENCE* se calcula la similitud con todos los macrobloques de la zona de búsqueda. La función de coste implementada es *Sum of Absolute Differences* (SAD).
- Etapa 3: Selección del macrobloque con el mínimo coste dando lugar a la actualización de las coordenadas del vector de movimiento.
- Etapa 4: Copia de los vectores de movimiento en memoria externa.

LISTADO 1 muestra el modelo C++ simplificado para Vivado HLS. Debido a las restricciones de espacio, sólo se muestran las estructuras de datos utilizadas por los parámetros de la función principal FSBM, las directivas `INTERFACE` no aparecen. El modelo está parametrizado gracias a la utilización de directivas *#define* que permiten elegir diferentes configuraciones de resolución, tamaño de macrobloque y zona de búsqueda.

Todas las etapas reciben y consumen los datos en forma de stream, implementados como canales FIFO, evitando la necesidad de utilizar memorias intermedias con estrategia de *ping-pong buffers* que requieren BRAMs. La única excepción a esta regla se produce entre la etapa 3 y 4 que utilizan dos *ping-pong buffer* para comunicar los valores de Vx y Vy. Así, para las diferentes resoluciones y un tamaño de ma-

[1] *Genlock Synhcronization* (página 39), Xilinx AXI Video Direct Memory Access v6.2 Product Guide (PG020).

crobloque de 16x16 (utilizado en este trabajo durante la fase experimental de obtención de resultados del IP FSBM), las dimensiones de las matrices son 40x30 (VGA), 80x45 (HD) y 120x68 (FHD) con elementos de 4 bits. Las memorias BRAMs utilizadas por Vivado HLS para realizar el mapeo de las variable son duplicadas para implementar la sincronización por medio de la estrategia *ping-pong buffer*.

LISTADO 1
MODELO HLS DE LA ARQUITECTURA DE FLUJO DE DATOS DEL IP FSBM.

```
1 #if defined(SA_16)
2 typedef ap_uint<4> MB_OFFSET_T;
3 #define SA 16
4 #endif
5
6 #if defined(VGA)
7 #define PIXELS_H 640
8 #define PIXELS_V 480
9 #define V_MB 30
10 #define H_MB 40
11  typedef ap_uint<6> MB_X_T;
12 #endif
13
14 #if defined (INPUT_BUS_WIDTH_16)
15 #define PIXELS_WORD 1
16 typedef hls::Mat<PIXELS_V, PIXELS_H, HLS_8UC4>
        YUV_IMAGE_T;
17 typedef hls::stream<ap_axiu<16,1,1,1>>
     AXI_STREAM;
18 #elif defined(INPUT_BUS_WIDTH_32)
19 #define PIXELS_WORD 2
20 typedef hls::Mat<PIXELS_V, PIXELS_H, HLS_8UC4>
     YUV_IMAGE_T;
21 typedef hls::stream<ap_axiu<32,1,1,1>>
     AXI_STREAM;
22 #elif defined(INPUT_BUS_WIDTH_64)
23 ...
24 #endif
25
26 #define NPIXELS_IMG (PIXELS_H*PIXELS_V)
27 void FSBM(AXI_STREAM& IMG_REF, AXI_STREAM&
     IMG_CURRENT, MB_OFFSET_T MOTION_X[V_MB][
     H_MB], MB_OFFSET_T MOTION_Y[V_MB][H_MB]){
28 ...
29     YUV_IMAGE_T CURRENT;
30     MB_OFFSET_T VX[H_MB];
31 #pragma HLS STREAM variable=VX off
32         MB_X_T MB_X[NPIXELS_IMG/PIXELS_WORD];
33 #pragma HLS STREAM variable=MB_X depth=2 dim=1
34
35 #pragma HLS dataflow
36         //Stage 0: Xilinx HLS Video data types
37         hls::AXIvideo2Mat(IMG_REF, REFERENCE);
38         hls::AXIvideo2Mat(IMG_CURRENT, CURRENT)
               ;
39         //Stage 1
40         extendBorders(CURRENT,CURRENT_EB);
41         orderMB(REFERENCE,MB_ORDERED);
42         //Stage 2
43         SAD(CURRENT_EB,MB_ORDERED,COSTS_SAD,
               MB_X,MB_Y,VX_OFF,VY_OFF);
44         //Stage 3
45         minCost(COSTS_SAD,VX_OFF,VY_OFF,MB_X,
               MB_Y,VX,VY);
46         //Stage 4
47         copy2Mem(VX,VY,MOTION_X,MOTION_Y);
48     return;
49 }
```

Debido al equilibrio entre las etapas, la profundidad de los canales de las FIFO han sido configurados con el menor valor posible (2 palabras), ayudando a moderar el uso de recursos en la FPGA. El ancho y la profundidad de los canales depende de la configuración de las interfaces Axi-Stream y de la resolución de las imágenes.

Respecto al consumo interno de recursos por parte de los módulos que implementan las diferentes etapas, el esfuerzo se ha centrado en utilizar una estrategia de ventana deslizante junto al empleo de *line buffers*. Esta estrategia permite en las etapas `orderMB` y `SAD` la utilización de *kernels* con un II=1 (intervalo de inicialización) aplicando la directiva `PIPELINE`.

A continuación, se analiza el detalle de la arquitectura para ayudar al lector a comprender el funcionamiento de cada etapa, los mecanismos de sincronización utilizados y la estrategia seguida para reducir la utilización de recursos.

III. Etapa 1: Adaptación del flujo de entrada

Esta etapa se encarga de la preparación de los frames de vídeo antes del cálculo de la función de coste SAD. Antes del comienzo de la etapa 1, se realiza una conversión de formato de los streams de entrada a una estructura de tipo Mat (tipo de datos básico de la librería para procesamiento de vídeo *hls_video* en Vivado HLS) utilizando las funciones *hls::AXIVideo2Mat* de Xilinx (lineas 37 y 38) del LISTADO 1.

El propósito general del algoritmo FSBM consiste en encontrar o estimar para cada macrobloque de referencia su relativa posición dentro de la zona de búsqueda establecida para cada macrobloque en el frame *CURRENT*. Los macrobloques situados en los bordes de los frames son un caso especial debido a que ciertas posiciones dentro de la zona de búsqueda no pueden ser obtenidas ya que se exceden los límites del frame. Esta problemática se puede resolver detectando cuando se está trabajando con información situada en el borde de los frames. Sin embargo, este enfoque genera un conjunto de sentencias condicionales *if-then-else* que añaden una lógica que rompe el estilo recomendado para HLS de bucle perfecto, provocando peores intervalos de iniciación al aplicar la primitiva `PIPELINE` y aumentando los periodos de ciclo de reloj.

En consecuencia, para lograr una implementación los más eficiente posible, en vez de emplear sentencias condicionales se ha optado por realizar una extensión de bordes que permita obtener los valores que exceden los límites de las dimensiones de los frames, completando la zona de búsqueda. Esta tarea es realizada por la función `extendBorders`. La figura Figura 3 representa gráficamente el resultado de este proceso. La técnica llevada a cabo consiste en replicar la información de los bordes, sin realizar interpolaciones, evitando aumentar la complejidad de los cálculos. Para este fin, es necesario la utilización de dos buffer con un tamaño de una línea de la imagen para guardar la información relativa a la primera línea y la ultima línea de la imagen. Para el resto de las líneas no es necesario almacenamiento intermedio ya que el orden de secuencia de extracción de los datos se corresponde con el orden de secuencia de rellenado de los stream de datos. Como salida de la función `extendBorders`, se genera un stream (*CURRENT_BE*) de 32-bit de ancho de palabra, que

empaqueta el valor de cuatro píxeles (8-bit de luminancia) por palabra.

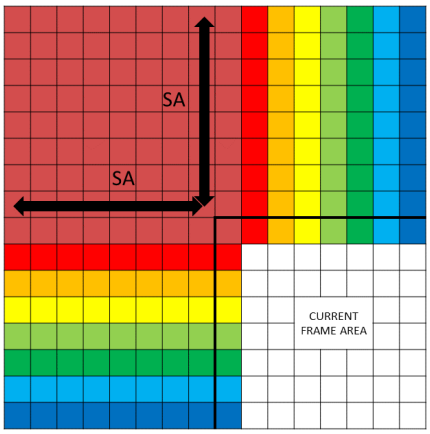


Fig. 3.Extensión del frame para completar el área de búsqueda en los bordes.

En paralelo a la extensión de bordes, la función `orderMB` realiza un reordenamiento de los píxeles del frame *REFERENCE*. La siguiente etapa espera recibir la secuencia ordenada de macrobloques de dicho frame, ya que permite un procesamiento secuencial de los píxeles del frame *CURRENT_BE*.

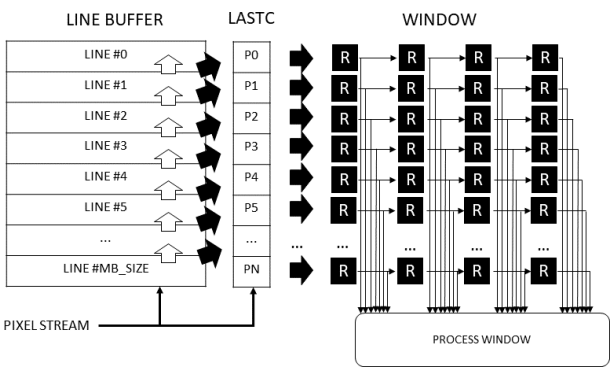


Fig. 4.Implementación de la técnica de ventana deslizante que se utiliza en las Etapas 1 y 2.

Para realizar el reordenamiento de los píxeles, se utiliza un enfoque de ventana deslizante con *line buffers*, reduciendo las necesidades de memoria BRAMs. La Figura 4 representa la arquitecura empleada en el IP FSBM.

El mecanismo modelado en HLS se muestra en LISTADO 2. La directiva `ARRAY_PARTITION` (línea 11) es utilizada para mapear las filas de la matriz que representa el line buffer en otras tanta memorias BRAM. En la líneas 9, esa misma directiva divide la ventana de procesamiento a nivel de registro. Esta asignación a recursos de la arquitectura permite el acceso concurrente, lo que posibilitará la planificación de varias operaciones de lectura/escritura en un mismo ciclo, permitiendo conseguir un intervalo de iniciación de un ciclo de reloj.

En `orderMB` se empaquetan macrobloques con una dimensión de NxN píxeles con un ancho de X-bit en una palabra de $NxNxX$-bit. En el prototipo desarrollado en este trabajo es $16x16x8 = 2048$-bit. El empaquetado se realiza siempre que los índices `row` y `col` sean múltiplos del tamaño de macrobloque (esquina inferior del macrobloque), cumpliéndose la condición de guarda. Por lo tanto, la palabra empaquetada representa el contenido de un MB en el frame de referencia.

LISTADO 2
PLANTILLA HLS PARA LA IMPLEMENTACIÓN DE LA TÉCNICA DE *line buffer* EMPLEADA.

```
1  typedef ap_uint<32> pixel_t;
2
3  #define W_SIZE ((MB_SIZE*MB_SIZE)/4)
4  #define W_MB (MB_SIZE/4)
5
6  pixel_t lastc[MB_SIZE];
7  pixel_t window[W_SIZE];
8  #pragma HLS ARRAY_PARTITION variable=window
       complete dim=0
9  pixel_t lineb[MB_SIZE][H_Lines/PIXELS_WORD];
10 #pragma HLS ARRAY_PARTITION variable=lineb
       complete dim=1
11
12 L1: for(row = 0; row < PIXELS_V; row++) {
13  L2: for(col = 0; col < PIXELS_H/PIXELS_WORD;
        col++) {
14 #pragma HLS PIPELINE II=1
15   // Line Buffer fill
16   for(idxMBSIZE_t i = 0; i < MB_SIZE-1; i++) {
17    lastc[i] = lineb[i][col] = lineb[i+1][col];
18   }
19   //Read Pixel Stream
20   PIXEL_STREAM >> p1;
21   pixel(7,0) = p1.val[0]; //Only Luminance
22 #if defined(INPUT_BUS_WIDTH_16)
23   PIXEL_STREAM >> p1;
24   pixel(15,8) = p1.val[0];
25   PIXEL_STREAM >> p1;
26   pixel(23,16) = p1.val[0];
27   PIXEL_STREAM >> p1;
28   pixel(32,24) = p1.val[0];
29 #endif
30 #if defined(INPUT_BUS_WIDTH_32)
31   pixel(15,8) = p1.val[2];
32   PIXEL_STREAM >> p1;
33   pixel(23,16) = p1.val[0];
34   pixel(31,24) = p1.val[2];
35 #endif
36 #if defined(INPUT_BUS_WIDTH_64)
37   pixel(31,24) = p1.val[2];
38   pixel(23,16) = p1.val[4];
39   pixel(31,24) = p1.val[6];
40 #endif
41   lastc[MB_SIZE-1] = lineb[MB_SIZE-1][col] =
         pixel;
42   //Shift Window
43   L3:for(idxMBSIZE_t i = 0; i < MB_SIZE; ii++)
44    L4:for( j = 0; j < (W_MB)-1; j++){
45     window[i*(W_MB)+j] = window[i*(W_MB)+j+1];
46    }
47    L5:for(idxMBSIZE_t i = 0; i < MB_SIZE; i++)
          {
48      window[i*(W_MB)+(W_MB)-1] = lastc[ii];
49    }
50    //if(condition){
51    //Process Windows: User Logic
52    //}
53 }
54 }
```

IV. Etapa 2: Cálculo de la función de coste

En esta etapa, cada macrobloque del frame *REFERENCE* es comparado con todos los macrobloques del segundo frame (*MB_ORDERED*) situados dentro de la zona de búsqueda. Se establece, por tanto, una relación de similitud entre macrobloques, basada en el valor calculado por la función de coste SAD *Sum of Absolute Differences*: cuanto menor sea el valor calculado por esta función mayor será la similitud entre macrobloques.

El objetivo es realizar todos los cómputos SAD con una latencia equivalente al número de píxeles del frame *CURRENT* con los bordes extendidos. Para este fin, al igual que en la anterior etapa, se utiliza una arquitectura de line buffer (ver LISTADO 2). Sin embargo, se introduce lógica para sincronizar la información de los streams `CURRENT_BE` y `MB_ORDERED`.

LISTADO 3
CÓDIGO HLS PARA LA ETAPA DE CÁLCULO DE LOS COSTES SAD.

```
1  ap_uint<1> load_MB = 0, fillMB = 0;
2  L1: for(row = 0; row < PIXELS_V; row++) {
3   L2: for(col = 0; col < PIXELS_H/PIXELS_WORD;
        col++) {
4  #pragma HLS PIPELINE II=1
5     if (load_MB) {
6      MBs[idxMBs_w++] = MB_ORDERED[idxMBs_stream
          ++];
7     load_MB = 0;
8     if (idxMBs_w == MB_H) {
9      idxMBs_w = 0;
10    }
11    //Stop consuming MBs
12    if (idxMBs_stream == MB_V*MB_H) {
13      fillMB_st = 1;
14    }
15   }
16   if ((col >= MB_SIZE) && ((col & (MB_SIZE-1))
        == MB_SIZE-1) && ((row & (MB_SIZE-1)) ==
        (MB_SIZE-1))) {
17    if (fillMB == 0) {
18      load_MB = 1;
19     }
20   }
21   //Line buffer architecture template
22   //...
23   //User logic
24   if(row >= MB_SIZE && col >= MB_SIZE){
25    idxMB_x = ((col-MB_SIZE)) >> 4;
26    idxMB_y = ((row-MB_SIZE)) >> 4;
27
28    MB_ref = MBs[idxMB_x];
29    sad_off = costSAD(window, MB_ref, &sadCost,
          (col & (MB_SIZE-1));
30    COSTS_SAD[idxCostsSAD] = sadCost;
31    MB_X[idxCostsSAD] = idxMB_x;
32    MB_Y[idxCostsSAD] = idxMB_y;
33    VX_OFF[idxCostsSAD] = ( (col + sad_off) & (
          MB_SIZE-1);
34    VY_OFF[idxCostsSAD_out++] = row & 0xF;
35   }
36  }
37 }
```

La Figura 5 muestra el mecanismo de sincronización y el patrón de consumo de los macrobloques. La imagen representa una versión simplificada de cómo solapa la información del frame *REFERENCE* (zona sombreada con gris claro) con la información del frame *CURRENT*. Cada celda representa a un subbloque de 8x8 píxeles y se asume en esta representación un tamaño de macrobloque de 16x16 píxeles, con un área de búsqueda de 8 píxeles. El momento en el que los macrobloques del frame *REFERENCE* son consumidos se representan con un diamante negro, a medida que se rellena el buffer `MBs` de un tamaño igual a una fila de macrobloques (pasos de a a d) en Figura 5). Las líneas 5-20 del LISTADO 3 implementan este comportamiento, mediante la activación de la bandera `load_MB` que será tenida en cuenta en la siguiente iteración.

Los valores de coste SAD se calculan en dos fases para cada macrobloque de referencia; la mitad

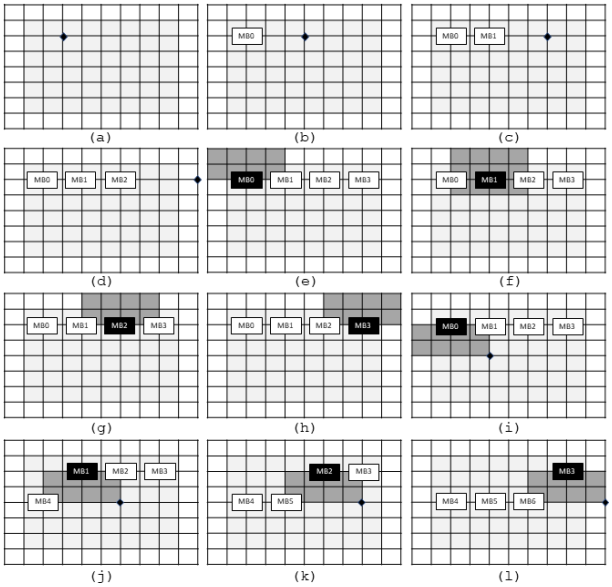


Fig. 5.Patrón de consumo de los macrobloques del frame de referencia y áreas de búsqueda.

superior (pasos e hasta h) y la parte inferior (pasos i hasta l). Después de la finalización de la segunda fase, los macrobloques en el buffer son remplazados por nuevos macrobloques, ya que no van a ser necesarios para los siguientes cálculos. Las celdas en color gris oscuro representan el área de búsqueda para un macrobloque de referencia (resaltado en color negro). Las líneas 25-28 en LISTADO 3) implementan la lógica de selección para el macrobloque de referencia.

A. *Función de coste*

La función `costSAD` es responsable de calcular la similitud entre macrobloques. Esta función recibe una nueva ventana de procesamiento cada ciclo de reloj, para poder mantener el rendimiento objetivo (un ciclo por píxel) de la ruta de datos. El stream `CURRENT_BE` tiene un ancho de palabra de 32 bits y en cada iteración se reciben cuatro componentes de luminancia (información del blanco y negro de los píxeles). Esto significa que, en cada iteración, cuatro valores tienen que ser calculados por la función `costSAD`. De otra forma, se perdería la información de tres columnas de píxeles motivado por el desplazamiento de la ventana deslizante.

La Figura 6 representa esta funcionalidad. La ventana se extiende una palabra (las últimas cuatro columnas resaltadas) para evitar la ya mencionada pérdida de píxeles. En HLS, la función `costSAD` (línea 30, LISTADO 3) implementa esta tarea. La lectura tanto de la ventana como del macrobloque de referencia (ambos mapeados en registros) tarda un ciclo de reloj en contemplarse. Después, por cada coste SAD calculado, se realiza la diferencia en valor absoluto de cada grupo de píxeles (4 en total) y se suman (línea 17, LISTADO 4). El balanceo automático de expresiones realizado por Vivado HLS produce la latencia mínima posible para el árbol de sumas.

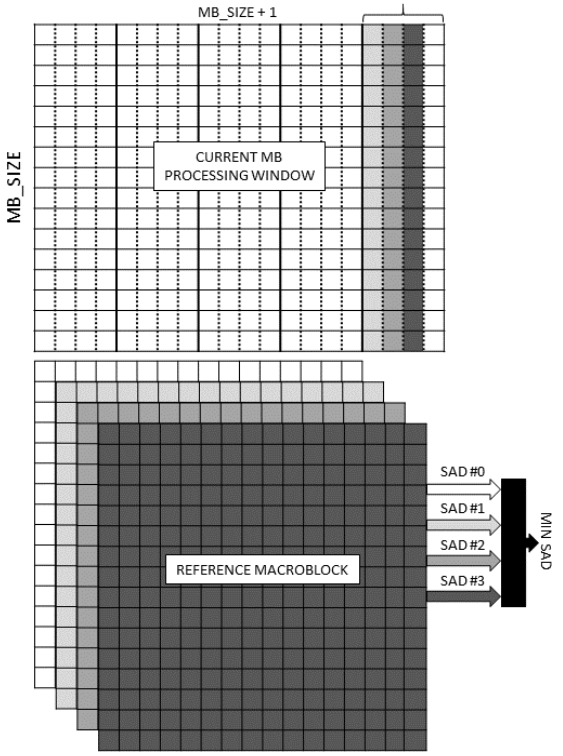


Fig. 6.Cálculo en paralelo de los valores de la función de coste SAD para la ventana de procesamiento extendida.

LISTADO 4
CÁLCULO DE COSTES SAD.

```
1  typedef ap_uint<2048> MBreg_t; //For 16x16 MB
       size
2
3  ap_uint<2> costSAD(pixel_t MB_curr[W_SIZE],
       MBreg_t MB_ref, COST_SAD_T *cost,
       MB_OFFSET_T offset){
4  #pragma HLS INLINE off
5   ap_uint<2> sad_select0, sad_select1,
        sad_select;
6
7   coste_MAD_i:for(i=0; i<MB_SIZE; i++){
8    coste_MAD_j: for(j=0; j<MB_SIZE/4; j++) {
9     p1R = MB_ref((i*MB_SIZE+j*4)*8+7,(i*MB_SIZE+
          j*4)*8);
10    p2R = MB_ref((i*MB_SIZE+j*4)*8+15,(i*MB_SIZE
          +j*4)*8+8);
11    //p3R, p4R
12
13    p1C = MB_curr[i*((MB_SIZE/4)+1)+j](7,0);
14    p2C = MB_curr[i*((MB_SIZE/4)+1)+j](15,8);
15    // p3C, p4C, p5C, p6C, p7C
16
17    SAD_0 += abs(p1R-p1C)+abs(p2-p2C)+abs(p3R-
          p3C)+abs(p4R-p4C);
18    //SAD_1, SAD_2, SAD_3
19   }
20  }
21 //1st Cycle
22  minSAD0 = SAD_0;
23  sad_select0 = 0;
24
25  if (SAD_1 < SAD_0)
26   minSAD0 = SAD_1;
27   sad_select0 = 1;
28  } else if (SAD_0 == SAD_1 {
29     if (offset < MB_SIZE/2) {
30      minSAD0 = SAD_1;
31      sad_select0 = 1;
32    }
33  }
34 //Idem for SAD_2 and SAD_3
35 //2nd Cycle
36  *costSAD = minSAD0;
37  sad_select = sad_select0;
38
39  if (minSAD1 < minSAD0) {
40   *costSAD = minSAD1;
41    sad_select = sad_select1;
42  } else if (minSAD0 == minSAD1){
43   if (offset < MB_SIZE/2) {
44     *costSAD = minSAD1;
45     sad_select = sad_select1;
46   }
47  }
48  return mb_select;
49 }
```

El parámetro `offset` proporcionado a `costSAD` representa la posición relativa del macrobloque de referencia respecto al área de búsqueda. Este parámetro se usa, en caso de igualdad entre valores SAD obtenidos, para seleccionar el macrobloque más próximo a la referencia (líneas 25-47, LISTADO 4). Esta decisión de diseño ha sido tomada debido a la alta tasa de frames que procesa el IP FSBM y a la naturaleza de la secuencia de vídeo (movimiento global), para poder detectar mejor pequeños desplazamientos entre frames. Como resultado, se selecciona el valor mínimo de SAD y el índice (`sad_select`) es retornado. De vuelta al módulo padre, la variable de selección de coste SAD sirve para ajustar la componente X del vector de movimiento (líneas 29 y 33, LISTADO 4).

V. Etapa 3: Selección del coste mínimo

Esta etapa recibe cinco flujos de datos de entrada, salidas de la anterior etapa SAD. El significado de estos flujos es el siguiente. Para un macrobloque de referencia (`MB_X, MB_Y`), un nuevo coste intermedio (`COST_SAD`) necesita ser procesado y comprobar si es el valor mínimo para ese macrobloque. En caso afirmativo, ese valor del vector de movimiento es actualizado guardando las componentes `VX_OFFSET` y `VY_OFFSET`.

LISTADO 5
CÓDIGO HLS PARA LA ETAPA MINCOST.

```
1  void minCost(COST_SAD_T COSTS_SAD[NPIXELS_IMG],
       MB_OFFSET_T VX_OFF[NPIXELS_IMG],
       MB_OFFSET_T VY_OFF[NPIXELS_IMG], idxMB_t
       MB_X[NPIXELS_IMG],idyMB_t MB_Y[NPIXELS_IMG
       ], MB_OFFSET_T VX[V_MB][H_MB], MB_OFFSET_T
       VY[V_MB][H_MB]) {
2  ...
3  ap_uint<4> distMX[MB_SIZE][MB_SIZE] = {...};
4
5  #pragma HLS ARRAY_PARTITION variable=distMX
       complete dim=0
6  COST_SAD_T COSTS_MIN[V_MB][H_MB];
7
8  //Init MIN_COSTS matric (MAX_COST_SAD)
9  ...
10 L1: for(idx = 0; idx < NPIXELS_IMG; idx++) {
11 #pragma HLS PIPELINE II=1
12  sad = COSTS_SAD[idx];
13  vx_offset = VX_OFF[idx];
14  vy_offset = VY_OFF[idx];
15  mbx = MB_X[idx];
16  mby = MB_Y[idx];
17  minSAD = MIN_COSTS[mby][mbx];
18
19  if (sad < minSAD) {
20   minSAD = sad;
21   MIN_COSTS[mby][mbx] = sad;
22   VX[mby][mbx] = vx_offset;
23   VY[mby][mbx] = vy_offset;
24  } else if(sad == minSAD){
25   vx_min = VX[mby][mbx];
26   vy_min = VY[mby][mbx];
27   dist_sad = distMX[vy_offset][vx_offset];
```

```
28    dist_min = distMX[vy_min][vx_min];
29
30    if (dist_sad < dist_min){
31     VX[mby][mbx] = vx_offset;
32     VY[mby][mbx] = vy_offset;
33    }
34   }
35  } // end of L1
36 }
```

La función `minCost` (ver LISTADO 5) inicializa la matriz `MIN_COSTS` con el máximo valor posible para coste SAD. Después, comienza el proceso de comparación entre el valor mínimo actual, guardado en `MIN_COSTS`, con al nuevo valor obtenido del flujo `COSTS_SAD`. En el caso de que estos valores sean iguales, se selecciona aquel macrobloque cuya distancia respecto al centro de la zona de búsqueda sea menor (lineas 23-31). Una ROM (variable `distMX`) es utilizada para almacenar los valores de las distancias precalculadas para reducir la latencia, sustituyendo una operación aritmética por un acceso a memoria.

En el final de esta etapa, los canales ping-pong buffer de `VX` y `VY` contendrán los valores calculados del vector de movimiento para el par de frames procesados. La función `copy2Mem` (Etapa 4) únicamente lee del *ping-pong buffer* y empaqueta los resultados del vector de movimiento en palabras con hasta 8 componentes (dependiendo del ancho de palabra seleccionado para la interfaz AXI-Stream).

VI. Resultados experimentales

El IP FSBM ha sido diseñado e implementado utilizando las herramientas de Xilinx Vivado HLS y Vivado en su versión 2016.4. El modelo y plataforma comentados en este trabajo fue migrado a la última versión disponible de las herramientas pero, sorprendentemente, los resultados de la síntesis HLS obtenidos por la versión 2018.3 fueron peores si los comparamos con los resultados de la versión 2016.4, mostrando un aumento considerable de los recursos del IP (especialmente el número de BRAMs).

El código C++ ha sido parametrizado completamente, permitiendo la posibilidad de variar la resolución de vídeo (VGA, High Definition, y Full HD), el ancho de palabra de la interfaz AXI-Stream (16, 32 y 64 bits) para acceder a la memoria donde se encuentran almacenados los frames y modificar la función `costSAD`, permitiendo la posibilidad de realizar 1, 2 o 4 (ver Figura 6 en la Sección IV) comparaciones a la vez con los valores de un macrobloque. La versión del componente utilizada para realizar las medidas de rendimiento tiene unos valores fijos para el tamaño de macrobloque ($N = 16$) y un valor del área de búsqueda ($p = 8$).

La Tabla I muestra las latencias (expresadas en ciclos de reloj) que resultan de procesar dos frames que entran en el cauce del IP FSBM. Dependiendo de la configuración de los parámetros, el rendimiento del IP se ve alterado por la modificación del ancho de palabra de la interfaz de memoria (e.j. la segunda fila en la Tabla I), y el intervalo de iniciación de la función `costSAD` (e.j. primera y tercera filas en la Tabla I). El intervalo de iniciación del cauce generado para la función `costSAD` es dos (II=2), debido a las dependencias de datos, sin importar su versión (1, 2 o 4), lo que retrasa el flujo de datos ya que el resto de los módulos cumplen con el objetivo (II = 1).

TABLA I
Valores de iniciación del *pipeline obtenido* (II) para diferentes configuraciones del IP FSBM (número de ciclos en cosimulación)

Conf.		Resolución de vídeo		
AXIS W	#SAD	VGA	HD	FHD
16,32,64	1	615765	1847206	4155822
16	2,4	325399	981699	2163569
32,64	2	308563	925210	2081650
32	4	170601	491961	1083441
64	4	154810	464802	1044850

El impacto en la utilización de los recursos para las diferentes síntesis realizadas (después del Place & Route) para la FPGA Xilinx ZC702 (ZedBoard), se recogen en la Tabla II. Debido a las restricciones de espacio en el documento, sólo se recoge un conjunto de configuraciones significativo para mostrar una serie de conclusiones. La primera es que cuanto mayor es la resolución mayor es la demanda de recursos; siendo esto un resultado esperado. Sin embargo, este incremento no sigue una relación lineal, sino que es un aumento moderado. La segunda, tiene que ver con la versión de la función `costSAD`, en este caso se produce un cambio significativo en los recursos utilizados. Esto es motivado por la cantidad de memoria extra que se necesita (mayor número de columnas en las ventanas de procesamiento) y por la lógica de adaptación de la ruta de datos, consumiendo un mayor un número de LUTs y FFs.

Respecto al rendimiento del diseño, en la Tabla III se muestra la tasa de frames por segundo (fps) procesados para cada configuración. Los resultados post-síntesis muestran una ligera variación en el período de reloj final para el IP FSBM según se modifican los parámetros del modelo. No obstante, se puede decir que el período promedio (8.69ns $\pm 0{,}19ns$) es constante, independientemente de la resolución de vídeo, la interfaz AXI-Stream y la versión de la función `costSAD` utilizada. La frecuencia de trabajo nominal ($\simeq$115 Mhz) es suficiente para realizar el procesamiento en tiempo real de un stream de vídeo, para las resoluciones VGA y HD con una tasa de refresco de 60Hz, gracias a la combinación de ciertos parámetros del IP que aseguran una óptima utilización de recursos en la FPGA. En cambio, para resoluciones Full HD, el componente no puede alcanzar la restricción del tiempo de píxel (esto es, 148.5Mhz/6.7ns para Full HD a 60 Hz). Por lo tanto, el diseñador debe encontrar una solución a esta restricción mediante la selección de una combinación de parámetros más agresiva, que permita cumplir el objetivo de realizar un procesamiento en tiempo real. Esta estrategia también se aplica en contextos donde el alto rendimiento es el principal objetivo de la aplicación.

TABLA II
UTILIZACIÓN DE RECURSOS (PLACA ZC702)

Conf.	LUT	FF	BRAM
	VGA		
16_SAD1	11647(21,89 %)	16780(15,77 %)	48,5(34,64 %)
32_SAD2	18186(34,18 %)	16824(15,81 %)	49,5(35,36 %)
64_SAD4	31424(59,07 %)	30820(28,97 %)	51,5(36,79 %)
	HD		
32_SAD2	18211(34,23 %)	17845(16,7 %)	50,5(36,07 %)
32_SAD4	30720(57,74 %)	24176(22,72 %)	52,5(37,50 %)
64_SAD4	31522(59,25 %)	31045(29,18 %)	52,5(37,50 %)
	FHD		
32_SAD2	18207(34,22 %)	19884(18,69 %)	53,5(38,21 %)
32_SAD4	30680(57,67 %)	26880(25,26 %)	55(39,29 %)
64_SAD4	31548(59,30 %)	31450(29,56 %)	55(39,29 %)

TABLA III
TASA NOMINAL DE FPS OBTENIDA (DISPOSITIVO ZC702, $\overline{T}$=8.69NS±2,23 %)

	Ancho palabra AXI-STREAM								
	16			32			64		
	Versión SAD			Versión SAD			Versión SAD		
	1	2	4	1	2	4	1	2	4
VGA	186	353	353	186	373	674	186	373	743
HD	62	117	117	74	124	234	74	124	247
FHD	27	53	53	41	55	106	41	55	110

VII. CONCLUSIONES

En este trabajo se ha presentado un IP que realiza una implementación de alto rendimiento del algoritmo de estimación de movimiento FSBM (*Full Search Block Matching*. La arquitectura propuesta se basa en una ruta de datos modelada con la herramienta Vivado HLS 2016.4. El modelo está parametrizado (resolución de vídeo, ancho de la interfaz de memoria, cálculos SAD en paralelo), permitiendo explorar fácilmente el espacio de soluciones.

El prototipo desarrollado sobre una placa ZC702 de Xilinx funciona a 115 Mhz, frecuencia suficiente para cumplir los requisitos temporales para VGA y HD 60Hz, con un consumo de recursos moderado. La implementación en tiempo real para Full HD necesita de un mayor ancho de banda a memoria y paralelizar los cálculos de coste SAD, alcanzando un máximo de 110 fps.

AGRADECIMIENTOS

Este trabajo ha sido financiado por el Ministerio de Economía y Competitividad de España bajo el proyecto `PLATINO (TEC2017-86722-C4-4-R)` y por la Junta de Comunidades de Castilla-La Mancha bajo el proyecto `SymbIoT (SBPLY/17/180501/000334)`.

REFERENCIAS

[1] S. Ghosh and A. Saha, "Speed-area optimized fpga implementation for full search block matching," in *2007 25th International Conference on Computer Design*, Oct 2007, pp. 13–18.

[2] H. Loukil, F. Ghozzi, A. Samet, M. A. Ben Ayed, and N. Masmoudi, "Hardware implementation of block matching algorithm with fpga technology," in *Proceedings. The 16th International Conference on Microelectronics, 2004. ICM 2004.*, Dec 2004, pp. 542–546.

[3] "Sad computation based on online arithmetic for motion estimation," *Microprocessors and Microsystems*, vol. 30, no. 5, pp. 250 – 258, 2006.

[4] A. Ryszko and K. Wiatr, "An assessment of fpga suitability for implementation of real-time motion estimation," in *Proceedings Euromicro Symposium on Digital Systems Design*, Sep. 2001, pp. 364–367.

[5] N. Roma and L. Sousa, "Efficient and configurable full-search block-matching processors," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 12, no. 12, pp. 1160–1167, Dec 2002.

[6] M. Mohammadzadeh, M. Eshghi, and M. M. Azadfar, "Parameterizable implementation of full search block matching algorithm using fpga for real-time applications," in *Proceedings of the Fifth IEEE International Caracas Conference on Devices, Circuits and Systems, 2004.*, vol. 1, Nov 2004, pp. 200–203.

[7] N. Roma, T. Dias, and L. Sousa, "Customisable core-based architectures for real-time motion estimation on fpgas," in *Field Programmable Logic and Application*, P. Y. K. Cheung and G. A. Constantinides, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 745–754.

FPGA Firmware description for IMaX+/SCIP Camera

Manuel Rodríguez[1], Eduardo Magdaleno[1], David Hernández[2], M. Balaguer Jiménez[2], Daniel Álvarez[2], Orozco Suarez David[2], A. C. López-Jimenez[2], del Toro Iniesta José Carlos[2], Ruiz Cobos Basilio[3] and Y. Katsukawa[4]

Abstract—**This paper describes the design and implementation of the camera firmware as part of IMaX+/SCIP project.**

The IMaX+/SCIP cameras are a new development for the chosen scientific sensor GPIXEL SENSE400 device family equipped with a FPGA.

The IMaX+/SCIP camera sensor has an active area of 2048 x 2048 pixels and it will reach a frame rate of up to 48 frames per seconds in Standard mode (STD). The firmware implemented in the FPGA is in charge of controlling the image sensor through a configuration interface, it controls the read out of the sensor data and the Coaxpress interface to communicate with the host device. The firmware has a Coaxpress communication interface up to 3.125Gbps.

The FPGA firmware allows the modification of several parameters such as the Region of Interest (RoI), the exposition time, trigger mode, single or continuous mode, the sensor gain and the black level offset adjust.

Keywords —**FPGA, Coaxpress, IMaX+/SCIP Camera, Embedded Sensor.**

I. INTRODUCCIÓN

THE IMaX+/SCIP sensor firmware (FW) was developed on Artix-7 FPGA, it controls and manages CoaXPress Interface (IF) communication in order to receive commands from the host device and send image data and control data (i.e. temperature, voltages, calibration data, etc.).

This develop belongs to the IMaX/Sunrise mission. The IMaX/Sunrise project was originally approved in 2002 by the National Program as a strategic step towards a technological demonstrator for the Solar Orbiter magnetograph (PHI). Since then, we have been funded with projects that cover the activities related to IMaX and to PHI projects. Our consortium, successfully have achieved the challenge the first aerospace instrument completely conceived, designed, developed, and flown by Spain. The IMaX has been flown twice in 2009 and 2013 aboard the Sunrise balloon-borne observatory. The IMaX+/SCIP instrument is a new improved version with respect to previous missions.

The main object of this paper is to show a summary of IMaX+/SCIP camera firmware. The design is based in the requirements for the design of the IMaX+/SCIP [1].

[1]Dpto. Ingeniería Industrial, Universidad de La Laguna, e-mail: mrvalido@ull.es

[2]Grupo de Física Solar, Inst. de Astrofísica de Andalucía (IAA-CSIC), e-mail: balaguer@iaa.es.

[3]Grupo de Física Solar, Inst. de Astrofísica de Canarias (IAC-CSIC), e-mail:brc@iac.es

[4]National Astronomical Observatory of Japan, e-mail: yukio.katsukawa@nao.ac.jp

II. IMAX+ /SCIP CAMERA FIRMWARE ARCHITECTURE

In the following figure, an overview of the IMaX+/SCIP camera FW architecture is presented.

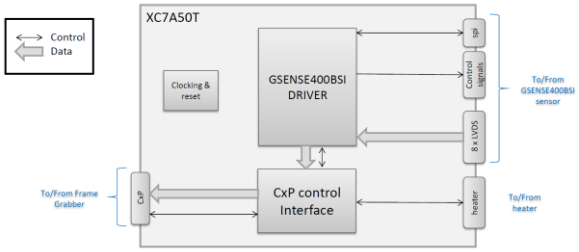


Fig. 1 Camera FW device Architecture overview

It is formed by two main blocks: GSENSE400BSI driver and CxP control Interface. The GSENSE400BSI driver is responsible for configuring, controlling and acquiring the sensor. Such a task implies to control an SPI interface, conversion the 8 LVDS serial data channels from camera to 8-12 bit parallel channels, channels calibrations task, generation the decoder that control the row logic address and timing control signals used to read out sensor data.

The CxP Control Interface is in charge of implement a high/low speed communication interface to send images to host and received command from host. This high/low speed interface is based on CoaxPress 3.0 standard. This block counts with an embedded microprocessor that acts as instruction decoder and permits:

1. To manage high speed and low speed communication with frame grabber or Host,
2. To run task on FW, (training sensor, read or write SPI sensor registers, reset of sensor, manage the heater system, set acquisition mode such as hardware/software triggered, continuous, or single frame acquisition. Also, from this embedded system the user can set up the sensor parameters, such us, integration time, set ROI, change training value, set image value for test

III. GSENSE400BSI DRIVER

The GSENSE400BSI driver, for the IMaX+/SCIP camera firmware development, is based on a project developed from Gpixel for the sensor Gsense400 [2]. It was developed in Verilog for a Spartan-6 xc6slx150-2fgg900 FPGA with a CameraLink interface.

We have migrated to the Artix 7 FPGA family from Xilinx [3] and modified it to the scientific requirements of the IMAX+/SCIP camera.

Also, to adapt it for IMaX+/SCIP cameras, several modifications have been performed. The main updates have been:

3. Change Spartan-6 xc6slx150-2fgg900 to Artix-7 xc7a50tcsg325-2
4. Export libraries and components from 6-series to 7 series Xilinx FPGAs
5. Migrate from ISE 13.1 to Vivado 2017.4
6. Change CameraLink interface to CoaxPress
7. Include MicroBlaze, XADC and heater control IPs
8. Include swapping module in order to arrange the image before transmission
9. Several minor changes

Figure 2 depicts a detail of the firmware block design. Green arrows are signals to/from CoaXPress control IF. Red arrows are signal to/from Gpixel sensor.

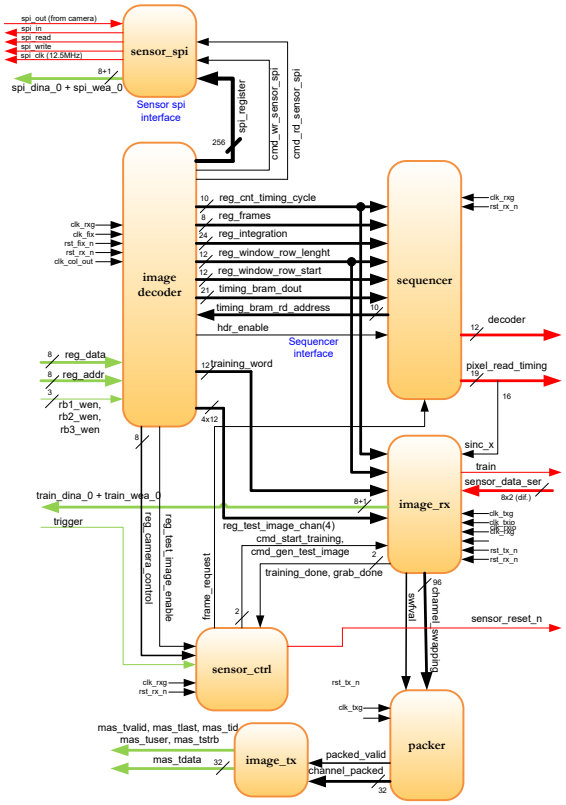


Fig. 2 Camera FW device Architecture overview

A. *Image Receiver Module*

The image receive module (Image_rx) parallelises to 12 bits width each serial LVDS data channels received from the sensor (*sensor_data_ser*). This module also performs the calibration process and generating the synchronization signals (*dval, fval, lval, grab_done, training_done*). Once the calibration, align series data with respect to the reading clock, is done, the module is ready to start the acquisition.

The acquisition of the data image is controlled by the Sensor Control Module through the input pulse *sync_x* signal. When it is set up to high, the image_rx module takes control of the acquisition. When acquisition process is done, *grab_done* output signal is setup to high. These data output, *data_out*, and data valid, *dval*, signals are sent to Transmitter module.

The *train* signal is connected to sensor in order to put this device in calibration mode. With train signal set to one, sensor sends the training word continuously. with this, we can check if the alignment of the data, pixel, are correct.

B. *Packer Module*

Packer module (figure 3) adapts the incoming 96-bit data (8 pixels) to 32-bit in order to send them via CoaXPress interface. This module implements three 32x1024 FIFO to change the clock domain from 25MHz to 75MHz and a 3-to-1 multiplexer. This multiplexer selects 31 to 0 bits, 63 to 32 bits and 95 to 64 bits for each incoming data.

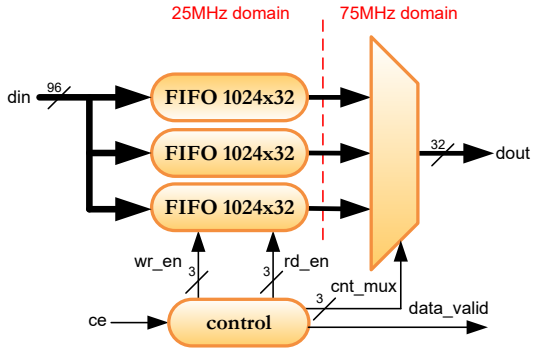


Fig. 3 Block diagram of packer module.

C. *Image Transmitter Module*

This module adapts data for sending to CoaxPress interface. *Swfval* data (figure 4) acts as chip enable. Row length registers signal, column start and columns length parameters are used to generate Axi Stream protocol signal *tlast*, *tvalid* and *tready* signals.

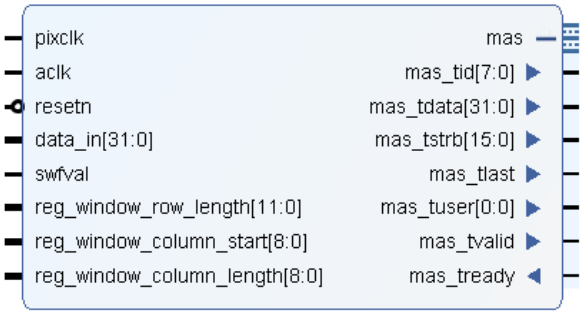


Fig. 4 Interface of Image transmitter module.

D. *Sensor Controller Module*

The sensor controller module is a 6-state FSM that controls the operating mode of the sensor. The 6 states are:

1. S_IDLE, remains inactive waiting for commands
2. S_RESET, resets the sensor, through the signal *sensor_reset_n* during 10.24µs using a counter (*cnt_reset*), complying with the specifications of the sensor reset.
3. S_TRAINING, activates training module (inside image_rx module) using the *cmd_start_training*

signal. The next state is the new state S_TRAINING2.

4. S_TRAINING2 set *cmd_start_training signal* to zero (this is a correction respect to the original). FSM returns to S_IDLE when it receives the *training_done* input signal from the image_rx the module.
5. S_GRAB, activates the *frame_req* signal. This signal is connected to the SEQUENCER_G400 module that sends the timing control signal to the sensor in order to capture the data image. The FSM returns to S_IDLE when it receives an assertive *grab_done* signal from image_rx module.
6. S_GRAB_TRIGGER, attends to hardware trigger incoming directly from CoaxPress IF. It asserts *frame_req* signal and returns to S_IDLE.

Commands to this FSM sensor controller are decoded through the 8-bit *reg_camera_control* register incoming from decode image module or the HW *trigger_in* signal from CoaxPress IF according to the values of Table 1

TABLA I

COMMAND CONTROL REGISTER

Bit	Command	FSM State	SW function
0	SPI read	--	Spi_read()
1	SPI write	--	Spi_write
2	Reset	s_RESET	resetSensor()
3	Training	S_TRAINI.	StarTrainig()
4	Frame Request	S_GRAB	requestFrame()
5	NOT USED	--	--
6	NOT USED	--	--
7	Go to IDLE	s_IDLE	--
trigger	HW Frame Req.	s_GRAB_TG	Not SW function

E. Sequencer Module

This module supplies the timing control signals and addresses of pixel rows directly to the sensor.

The sequencer module consists basically of a set of counters and three FSMs that are responsible for sending the timing control signals together with the direction of the pixels row of the sensor that will be read or reset. Timing control signals are obtained from a memory of 513x21 allocated in the image decode module. The content of this memory depends on the operating mode of the sensor (STD or HDR). The FSMs are:

1. *fsm_timing*, to manage the timing slots for each row read/reset operation
2. *fsm_read_decoder*, to decode the sensor row address for read operation
3. *fsm_rst_decoder*, to decode the sensor row address for reset operation.

FSMs, counters and the synchronization signal *sync_x*, which goes to the module image_rx, behave according to the registers that come from the module register_banks_interface (*reg_integration, reg_cnt_timing_cycle, reg_window_row_start, reg_window_row_ length, reg_frames and hdr_enable*). These registers determine the mode of operation of the sensor (STD or HDR), the number of images to be captured, the integration time, the size in rows and the read start row. Output ports, decoder and *pixel_read_timing*, of this module are connected to the sensor, according to the specifications [2].

F. Image Decoder Module

The image decoder module is directly connected to MicroBlaze through an Axi lite IP.

This module includes three memory banks (RB1, RB2 and RB3) to store configuration parameters, and a 513x21 BRAM to store timing control signals. The image decoder module receives software commands through *rb1_wen, rb2_wen, rb3_wen, reg_addr* and *reg_data* signals. These data are written in the corresponding memory bank and the rest of firmware operates according to the received values.

G. Sensor SPI Module

This module can read and write registers map in the sensor using SPI protocol.

There are two FSMs implemented, *fsm_gen_spi_clk* and *fsm_sensor_spi*. First of them, generates *spi_clk* (12.5MHz) when a communication between the sensor and the firmware is required. The second FSM is used to send/receive data using *spi_out* and *spi_in* ports. In order to write the sensor register map, the SPI registers (from image_decoder module) are de-serialized and transmitted.

IV. COAXPRESS CONTROL INTERFACE

This module is in charge of performing the control of the CoaXPress interface. A block diagram is presented in the following figure.

IMaX+/SCIP camera firmware has a Xilinx block design, based in Axi lite bus architecture. The Microblaze, a basic Xilinx soft processor module, manage FW hardware IPs cores: the heater control, SPI, Axi to camera register, the sensor training and the XADC controller and CoaXPress device with its CoaxPress Transceiver controller. The latter, is connected to FPGA GTP transceiver interface.

All these IPs have associated a local CoaxPress Address to be commanded by host frame grabber. This is done through the software decode, in the main firmware program.

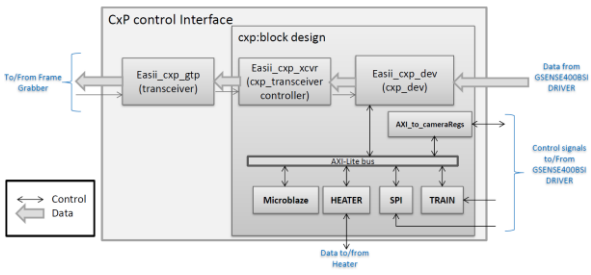


Fig. 5 Coaxpress Control Interface block diagram

A. *Heater Controller IP.*

This IP core has been designed according to the specifications [1]. The thermal control is performed by means of 4 commands that are mapped to the Microblaze space address. These commands are sent from the frame grabber host and the firmware software decodes them into local actions, i.e. setting set point value, Enable/disable heater and read heater status.

This thermal control is carried out activated /deactivated by setting the signal called *En_peltier_fpga*.

The desired temperature for the heater control is also received from the CxP host by a command the heater control IP core configures via SPI the Texas Instruments DAC8311IDCKT. The SPI clock is limited to 1MHz max due to path restrictions. The IP core has a timer configured to periodically refresh the DAC to prevent it from losing its configuration (every second) the value of the desired setpoint.

This IP core it's an Axi lite wrapper of a main functionality module named U_top_DAC8311. The figure below shows the architecture and the ports interface. The submodule DAC_TIMER generate a 1 Mhz clock signal used to communication protocol of DAC8311, also, generate a 1 second periodical signal, CS. It is used refresh the set point value into DAC periodically. The other component its par2ser_16bit, which converts the parallel set point value to 16 bit serial and *SCLK* and *nSYNC* signals necessary to communicate with DAC. In the schematic, also, there are several register to stores the set point input value, heater control commands and heater status register. The module Serial_I_P_O_for debug it is used for debug purpose only

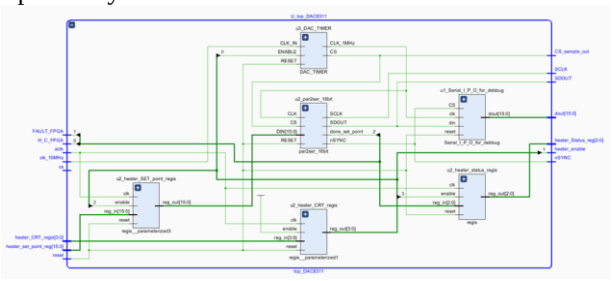


Fig. 6. Heater Control Architecture block diagram

The camera's FW communicates with the IP through 3 registers located in the address space of the Microblaze system. Through them we can read the status and set the temperature of the thermal control system. All these operations are governed by the CoaXPress host sending commands. To carry out these operations, we have designed a driver in c language.

B. *SPI IP.*

This IP was designed to read data from the camera configuration registers and the sensor internal temperature sensor. This operation is commanded by host frame grabber.

From the hardware point of view, Gpixel sensor module has 32 8 bits registers or 256 configurations bits. This register can be acceded to read through SPI interface. The architecture of this is IP is based in a dual real port BlockRam module. It is wrapped with Axi lite interface on the Microblaze side and a 8 data bit on the SPI sensor module. Figure 23 show the IP interface.

C. *Sensor Training IP.*

This module was designed to manage the sensor training process.

The training of Gpixel sensor is performing at the start firmware software. It is repeat up to all data channel are calibrated. To avoid infinite loops in firmware software the training operation are limited to 8 iterations and after this, the core read the training sensor status register.

D. *CoaxPress Device IP.*

This IP was developed by Easii-IC, French Company and has associated a reference design DOC [4] that was used to implement the CoaxPress Control interface. CoaxPress IP interface video sensor data with host frame grabber using a single coaxial cable. In Our case the link it is set to 3.125Gbps.

E. *Axi to Camera Registers IP.*

This IP manage the write operations in the configuration registers Bank RB1, RB2 and RB3 build in the decoder module.

V. CLOCKING MODULE

This is a new module regarding the original project. There are two clocking wizard IP inside this module. In the first of them, clk_25M clock (OSC) is used to generate the following buffered clock signals: *clk_fix* (25MHz), *clk_rxg* (25MHz), *clk_main* (25MHz), clk_tx75 (75MHz) and clk_200_idelay_ctrl (200MHz). In the second one, *clk_col_out* clock (clk_pix incoming from sensor device, 25MHz) is used to generate the following buffered clock signals: *clk_lvds_sdr_in* (300 MHz), *clk_rxio* (50MHz) and *clk_pix_buf* (25MHz). *Clk_lvds_sdr* signal clock is connecting to sensor device but it is not used.

Also, this module is used to buffer incoming *rst_n* and supply *rst_fix_n*, *rst_rx_n* and *rst_tx_n* signals.

VI. MICROBLAZE FIRMWARE SOFTWARE BLOCK

The software that controls camera sensor is developed using Xilinx SDK IDE in C. The main tasks of this software is to initialize everything necessary (registers, CoaxPress communication, etc.) and to create a bridge between the host and the sensor device, to control the acquisition.

The software interacts with the HW firmware of the sensor through of 3 registers banks. In the initial stage the sensor is reset the firmware software set all default values into register banks and send RB2 to SPI sensor register.

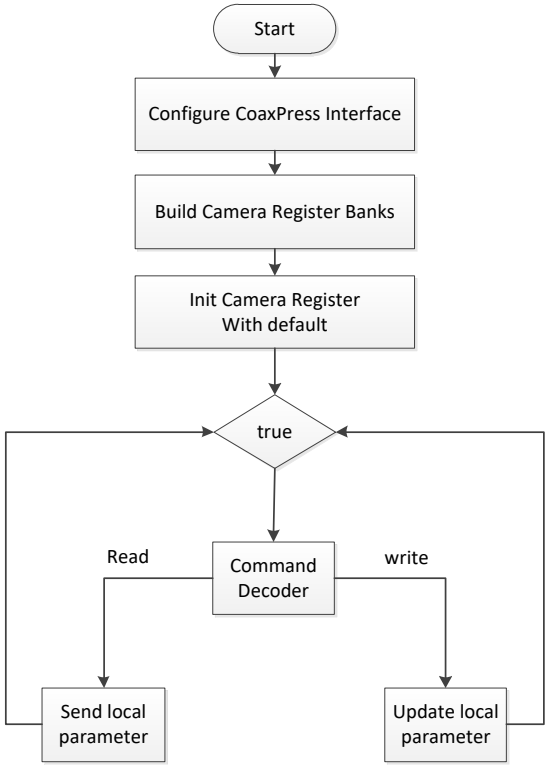


Fig. 7 Main loop of Microblaze embedded software

Figure 7 depicts the flow of the main code. At the beginning, the firmware starts with the initialization process, connecting the CoaXPress interface to the host frame grabber. In this stage, there is a discovering stage where the communication speed is setting (3,125Gbps) also the device send xml file with default values) [5]. After this, the FW software creates/initializes the registers banks RB1, RB2 and RB3 with the default values, the sensor registers are update too.

Once this process has been finished, the main code stays in a commands decoder loop, waiting for a request from the host. When a request is detected, this is decoded, (reading or writing), and then return or modify the value of a register depending on the request.

In *Configure CoaXPress Interface* stage, several processes are executed in this stage, such as, create the interruption table, star discovered and send XML file.

In *Build Camera Register bank* the RB1, RB2 and RB3 structure register banks are created and initialized with default values. In order to reference the camera physical addresses, it is necessary to create an abstract layer (virtual memory) that assigns each register of the camera to a virtual address (LUT). This helps to reference registers in a more natural way without needing to know their internal structure. This process is made inside the "BuildCameraRegistersBanks()" function, and it is called in the main only once. An example of writing camera value of register is describing below.

VII. Results and Conclusions.

The entire system was tested satisfactorily. The camera and its firmware, was connected to a commercial CoaXlink frame grabber and also a framegrabber design and develop for this project. Both, compatible with standard GenICam protocol [5].

Figure 9 shows a 2048 * 2048 capture of a test pattern that we designed for that purpose.

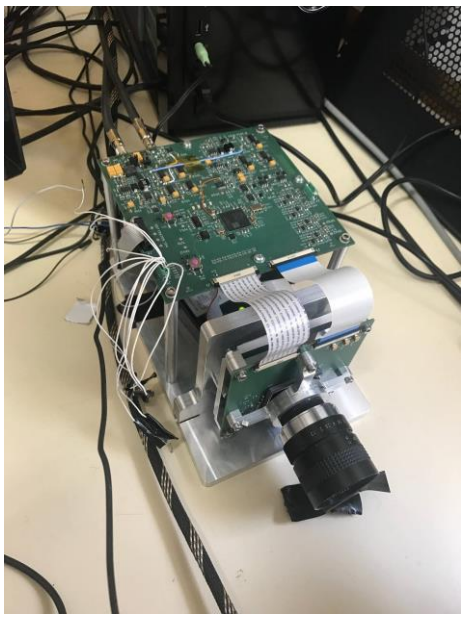


Fig. 8 Picture of hardware prototype of IMAX+/SCIP

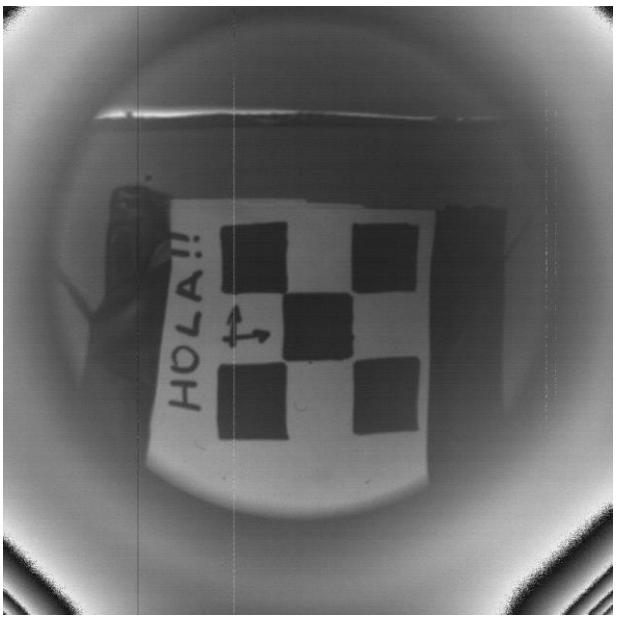


Fig. 9 2048*2048 *test image*, captured by IMAX+/SCIP camera.

ACKNOWLEDGMENTS

This work has been funded through the National Program: ESP2016-77548-C5.

REFERENCES

[1] IMAX + Camera Requirements Document.Solar Group, IAA (Instituto Astrofísica de Andalucía.
[2] 4 Megapixels Scientific CMOS Image Sensor. Data sheet GPIXEL400BSI.
[3] 7 Series FPGAs Configuration User Guide. Ug470 Xilinx reference.
[4] CoaxPress Device IP Specification. IC/130206. Hard Soft Interface Document (EASii IC).
[5] Generic Interface for Cameras standard, GenICam Package Version 2018.06.

Estrategia multi-hilo para la mitigación de fallos software inducidos por radiación en sistemas empotrados carentes de sistema operativo

Alejandro Serrano-Cases[1], Leonardo Maria Reyneri [2], Sergio Cuenca-Asensi[1], Antonio Mártinez-Álvarez[1]

Resumen— **Este artículo presenta una técnica software para la protección de datos frente a fallos inducidos por radiación basada en una estrategia multi-hilo con triplicación de datos en memoria. Nuestra estrategia hace uso tanto de técnicas de triplicación de datos como de técnicas de triplicación y duplicación de flujos de instrucciones con el fin de mejorar la fiabilidad y correcto funcionamiento del sistema. Los resultados obtenidos muestran que a pesar de ser consideradas técnicas que implican un sobrecoste evidente en el número de instrucciones y memoria empleada, la paralelización de las técnicas de triplicación de flujos obtiene mejores resultados que las técnicas duplicación, llegándose incluso a mejorar los tiempos de recuperación frente a errores. Además, este estudio pone de manifiesto la importancia de la protección de los datos que residen en memoria, ya que la triplicación de flujos de instrucciones no es suficiente para mejorar por sí sola la fiabilidad total del sistema.**

Palabras clave— **Tolerancia a fallos, fiabilidad, replicación, sistemas empotrados,** *bare-metal*, *lock-step*, **SEU, fallos inducidos por radiación,** *soft errors*

I. Introducción

La continua miniaturización de los dispositivos electrónicos hace que éstos sean cada vez más vulnerables a los efectos de la radiación y, como consecuencia, sean menos fiables. La radiación afecta a los distintos componentes de los dispositivos, alterando sus propiedades y en consecuencia el comportamiento esperado del mismo. Estas alteraciones pueden llegar a generar desde resultados erróneos en los circuitos, hasta la destrucción de parte de los mismos, dejando inoperable el dispositivo [1]. Como consecuencia, los desarrolladores de sistemas críticos, tales como satélites, sistemas de aviación o conducción autónoma, tienen que asegurar sus diseños con pruebas de verificación exhaustivas, donde se verifique que los dispositivos electrónicos continúen operando ante este tipo de incidencias. En este sentido, la literatura ofrece una amplia cantidad de técnicas para hacer frente a este tipo de problemas. Atendiendo a dónde se apliquen, estas técnicas las podemos dividir en hardware, software e híbridas. La mayoría de estas técnicas se basan en replicación de componentes para detectar y/o corregir fallos inducidos por radiación, siendo la Redundancia Triple Modular (del inglés TMR) la técnica de mitigación más extendida.

Las técnicas hardware están basadas mayoritariamente en replicación de registros, memorias o CPUs enteras. Aunque estas últimas resulten ser muy efectivas, tienen un alto coste económico además de largos tiempos de desarrollo encaminados a un producto de uso específico. Un ejemplo de estas técnicas son las basadas en *Dual-redundant Core Lock-Step* (DCLS) y *Triple Core Lock-Step* (TCLS)[2], que replican los diferentes procesadores y comparan el estado de los mismos ciclo a ciclo para detectar discrepancias en la ejecución de una aplicación.

Las técnicas de replicación software están encaminadas a habilitar de forma programática una computación fiable en dispositivos de uso general, como los microcontroladores y dispositivos empotrados. En ellas se introduce la replicación a distintos niveles de los componentes software que componen un programa (funciones/métodos, bucles, instrucciones ensamblador, etc.), o del programa completo en su totalidad. Aunque su desarrollo es más rápido y menos costoso que las técnicas hardware, estas técnicas presentan sobrecostes en la ejecución y en el uso de recursos que hay que valorar y tener siempre en consideración. Un ejemplo de técnica software es la llamada *Trikaja*[3], donde el programa a proteger se ejecuta dos veces en ausencia de error y una tercera vez si en las salidas de las dos réplicas precedentes se ha detectado alguna discrepancia al realizar la comparación. Otras técnicas software de reciente aparición tratan de buscar la mejor cadena de compilación, para una aplicación dada, que maximice la tolerancia a fallos de dicha aplicación [4] y [5]. Este efecto de mejora de la fiabilidad como consecuencia de compilar de forma distinta no está basado en la instrumentación del código fuente para introducir replicación y, por tanto, no se replica ninguna parte del programa.

Por último, las técnicas híbridas, intentan integrar y mejorar los resultados obtenidos por las técnicas anteriores, mitigando algunas de sus deficiencias. Un ejemplo de técnica híbrida se encuentra en [6], donde los autores muestran una metodología de

[1]Departamento de Tecnología Informática y Computación - (DTIC), Universidad Alicante, e-mail: {aserrano,sergio,amartinez@dtic.ua.es}

[2]Dipartimento di Elettronica e Telecomunicazioni (DET), Politecnico di Torino, email:leonardo.reyneri@polito.it

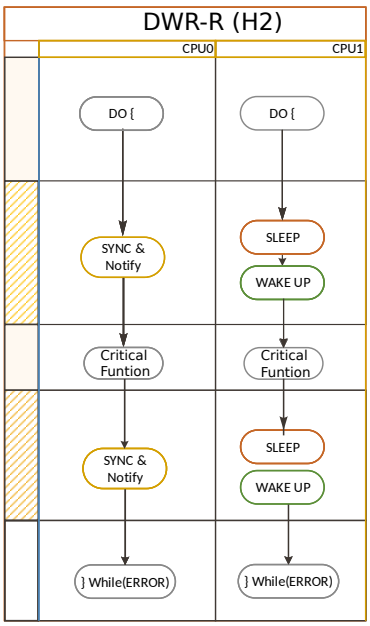


Fig. 1. Esquema de funcionamiento del DWR-R

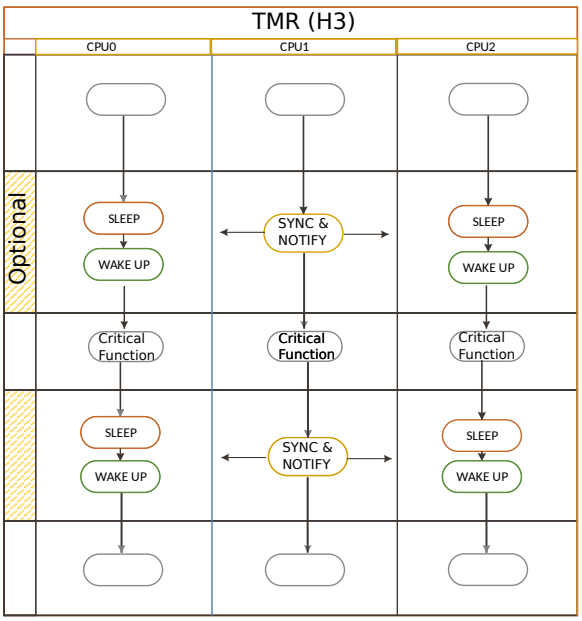


Fig. 2. Esquema de funcionamiento del TMR

co-diseño y una infraestructura de endurecimiento que permite una fácil exploración del espacio de diseño entre las técnicas de mitigación puramente hardware y puramente software.

Los últimos avances tecnológicos en los procesos de fabricación electrónica posibilitan que los sistemas multi-núcleo sean ya una realidad ubicua en computación empotrada. Estos sistemas pueden ayudar a acelerar la computación destinada a mitigar o detectar fallos software producidos en ambientes hostiles. Sin embargo, muchas de las técnicas de protección software existentes desaprovechan los nuevos recursos de computación, al no estar diseñadas a tal efecto. Investigaciones recientes apuntan hacia el aprovechamiento de estos componentes mediante la paralelización de las tareas de replicación y comprobación del procesamiento para ganar en eficiencia. En este sentido, trabajos como [7], [8] y [9] utilizan y evalúan bibliotecas de alto nivel como *OpenMP* y *PThread* para generar la redundancia software, o [10], donde se utiliza una modificación de *PThreads* (*RedThreads*) para la generar redundancia orientada a la fiabilidad. No obstante, todos ellos ponen de relevancia un claro sobrecoste en el rendimiento debido a la complejidad introducida por las propias bibliotecas. Otra variable a tener en cuenta es el efecto del uso de un sistema operativo, el cual se puede considerar otra fuente de posible errores al no estar protegido[11] y suponer un sobrecoste que a veces es inadmisible debido a la gran cantidad de tareas que llega a desarrollar. En este sentido, trabajos recientes como [12], [13] y [14] han desarrollado técnicas multi-hilo en entornos sin sistema operativo con resultados prometedores. En [12], los autores muestran un técnica multi-hilo de triplicación de datos instrumentando el ensamblador, obteniendo una mejora en la cobertura frente a fallos de 26× y unos sobrecostes de ejecución cercanos al 8× de media. En [13] se muestra una técnica de duplicación con re-ejecución donde se observa un claro sobrecoste en las rutinas de restauración, además de otra técnica de triplicación donde los sobrecostes en eficiencia son menores que las técnicas de triplicación clásicas. En [14], la técnica de triplicación se evalúa en mayor profundidad, mostrando claramente una influencia positiva en la reducción de fallos en los registros.

El presente trabajo por su parte, extiende los esfuerzos ya realizados en [13] y [14], añadiendo a las técnicas de triplicación y duplicación de flujos de instrucciones propuestas la triplicación de los datos residentes en memoria. Además, esta técnica ofrece la posibilidad de distribuir las réplicas entre distintos componentes, evitando así que un fallo afecte a más de una réplica.

II. Metodología

Nuestra propuesta hace uso de dos técnicas de corrección de errores basadas en redundancia de software que usa el concepto de esfera de replicación[15] (del inglés SoR), *Duplication With Comparison with Re-Execution* (DWR-R) y *Triple Modular Redundancy* (TMR), a las que se ha añadido protección mediante triplicación (TMR) a los datos que residen en memoria durante la ejecución del programa.

A. DWR-R

La figura 1 muestra el esquema de funcionamiento de la técnica DWR-R paralelizada, que está basada en dos réplicas que se ejecutan concurrentemente en dos procesadores (CPU0 y CPU1) hasta una primera zona de sincronización, o en nuestra terminología, hasta la entrada a la esfera de replicación. En este punto se procede a crear un punto de restauración guardándose el estado de las variables de entrada a una zona protegida o contexto. Posteriormente, la

ejecución del programa entra en la sección protegida (SoR) y realiza los cálculos de la función o bloque de programa que nos interesa proteger, hasta llegar al segundo punto de sincronización o salida de la esfera de replicación. En este punto se decide si volver al primer punto de sincronía y restaurar el contexto para volver a re-ejecutar los cálculos en caso de error, o si por el contrario, se continua el flujo normal del programa porque los resultados son correctos.

B. TMR

Por otro parte, la figura 2 muestra el funcionamiento de la técnica TMR paralelizada, que está basada en tres réplicas que se ejecutan concurrentemente sobre distintas unidades de procesamiento hasta una primera zona de sincronización o de entrada a la esfera de replicación. Nótese que la sincronización de las variables de entrada en este punto es opcional. Posteriormente se procede a la ejecución del código de la sección protegida hasta llegar a una última zona de sincronización o salida de la esfera de replicación. En esta zona, a diferencia de la técnica anterior, el valor correcto se obtiene por votación mayoritaria y no necesita re-ejecución antes de proseguir con el flujo normal del programa.

C. Triplicación de Datos

Las dos técnicas anteriores consiguen eliminar parte de los fallos que se propagan al sistema porque no tienen en cuenta la persistencia de los datos en memoria, donde la presencia de fallos corrompe los datos sin posibilidad de recuperación. Por ello, este trabajo añade la protección de datos de una aplicación en función de su localización en distintas secciones de memoria. Por ejemplo, se pueden operar de forma distinta los datos residentes en secciones de solo lectura (*rodata*), secciones de datos inicializados (*data*) y secciones de datos no inicializados (*bss*), mediante la triplicación de la información.

Para asegurar que los fallos que afecten a un componente del dispositivo no interfieran en el cómputo y se pueda continuar con el procesamiento, se han habilitado distintas zonas de memoria para cada una de las réplicas, de forma que sea posible asegurar que sean alojadas en distintas memorias del dispositivo. Para conseguir dicho fin, se han alterado los mapas de memoria que utiliza el compilador para dicha plataforma durante la fase de enlazado del ejecutable. Para ello, se han creado y *mapeado* nuevas zonas o secciones de memoria compatibles con el espacio de direccionamiento proporcionado por los distintos componentes de memoria del dispositivo. De esta forma se asegura que el almacenamiento es independiente para cada una de las réplicas. Es decir, cada réplica se aloja en un componente o sección de memoria distinta.

D. Endurecimiento Automático mediante Anotaciones de Código

Las técnicas mencionadas anteriormente se han aplicado siguiendo una estrategia de anotaciones al código fuente que mantiene un nivel de intrusión mínimo. Esto quiere decir que el código a endurecer ha de incluir únicamente los cambios imprescindibles para su endurecimiento, y por tanto, no se precisa de una reescritura completa del código para adaptarse a las diferentes técnicas de endurecimiento anteriormente descritas. Es más, este enfoque, basado en anotaciones, está diseñado para restaurar la funcionalidad del algoritmo original cuando no se activa ningún tipo de endurecimiento.

Para ello, dependiendo del elemento a proteger (variable o región de código) este artículo propone utilizar tres aproximaciones distintas: 1) aplicación de las técnicas de replicación de flujos, 2) replicación de datos globales y 3) replicación de variables automáticas.

Un ejemplo aplicación de cada una de las tres aproximaciones se puede observar en la figura 3, donde se ha modificado la declaración de las variables y se ha anotado el bloque a proteger.

La primera aproximación (replicación de flujos), se obtiene iniciando el mismo programa en los diferentes núcleos de la plataforma seleccionada. Así, dependiendo del número de núcleos que se activan se aplica una de las estrategias de replicación anteriormente mencionada (*H2-DWR-R* o *H3-TMR* para distintos núcleos y *H2T1-DWR-R* o *H3T1-TMR* para las versiones de un solo núcleo o clásicas). En la figura 3 se observa cómo las variables `entryVar` y `autoVar` se sincronizan con la primera llamada a la macro `SYNC` y `outVar` en la segunda llamada de a la macro `SYNC`. En estos puntos es donde el cómputo del programa diverge y cambia su comportamiento normal dependiendo de la técnica de protección y del número de núcleos del procesador.

La segunda aproximación (replicación de variables globales) se ha llevado a cabo con la sustitución de la declaración de cada una de las variables por macros del lenguaje *C*. De esta forma, dependiendo del nivel de endurecimiento de los datos seleccionados, la declaración de las variables se triplica o se deja como en la versión original. En la figura 3, se observa cómo las macros `RO_HARD`, `DATA_HARD`, `BSS_HARD` se encargan de realizar la triplicación de las variables cuando el símbolo `D3` está definido. Seguidamente, se genera una segunda definición con el nombre de la variable y la macro (`PTR`), con el fin de no modificar el programa y hacer transparente el uso de la variable triplicada con su nombre original. Esta declaración, *per se*, no tiene efecto alguno hasta que se acceda a una zona de replicación donde se conoce el identificador de la replica en ejecución y la declaración se completa con las declaraciones de (`FC_HARD` y `FN_HARD`). Al final, de forma automática, los nombres originales de las variables apuntan en todo momento a la réplica que

les corresponde.

Por último, la tercera aproximación (replicación de variables automáticas), se realiza de forma transparente para el usuario debido a la separación de pilas en las técnicas multi-hilo. En la figura 3 se observa la variable `autoVar`, que se encuentra en la sección de `stack` de cada uno de los núcleos de procesamiento y es independiente del resto de réplicas. Por ello, en la primera sincronización se ha procedido a verificar su contenido con los otros procesos. En caso de tener un solo núcleo de procesamiento, se fuerza a que esta variable tenga un carácter de solo lectura, puesto que las ejecuciones sucesivas (las réplicas) tienen que tener el mismo valor de entrada para poder efectuar el mismo cálculo sobre los datos replicados.

```
#define D3        //Data TMR
// #define H2     //Thread Instruction DWR-R
#define H3        //Thread Instruction TMR
// #define H2T1   //Classic DWR-R
// #define H3T1   //Classic TMR

#include "MTH.h" //Macro definitions

///
///  Original definitions of variables:
///  constVar, initVar and noInitVar.
///

// const int constVar = 3;
// float initVar  = 5.0;
// char noInitVar;

///
///  Anotated versions of variables:
///  constVar, initVar and noInitVar.
///

RO_HARD(int, constVar, 3)
#define constVar  PTR(constVar)

DATA_HARD(float, initVar, 5.)
#define initVar   PTR(initVar)

BSS_HARD(char, noInitVar)
#define noInitVar PTR(noInitVar)


void foo(int autoVar){
    int entryVar;
    int outVar;
    ...
    SYNC(entryVar, autoVar, ...)
    FC_HARD(int, constVar)
    FN_HARD(float, initVar)
    FN_HARD(char, noInitVar)
        ...
        ///
        ///  SoR section
        ///
        int fooVar = 4;
        noInitVar = 3 * constVar << fooVar;
        outVar = noInitVar * DataVar;
        ...
    SYNC(outVar)
    ...
}
```

Fig. 3. Ejemplo de la instrumentación de código propuesta para endurecer variables usando anotaciones. En el ejemplo se observa cómo se han anotado las variables globales (`constVar`, `initVar` y `noInitVar`) para su triplicación según la técnica propuesta *D3*. También se muestra la notación para proteger la sección de replicación indicada con la técnica TMR (*H3*), anotando las variables sobre las que se realizará la sincronización (`entryVar`, `autoVar` y `outVar`).

Esta estrategia de anotaciones presenta las siguientes limitaciones: el espacio compartido del montículo (*heap*) y el uso de variables con el operador de igualdad. En el primer caso, el *heap* es una zona de memoria donde la función `malloc` aloja las variables dinámicas del programa. Aunque, el uso de memoria dinámica no es recomendado en programas críticos, éste presenta la dificultad de que no puede ser replicado en diferentes secciones de memoria, además no se puede asegurar un funcionamiento *aislado* entre los núcleos de procesamiento. Por otro lado, los tipos de variables que se pretendan usar para sincronizar deben soportar el operador de igualdad para determinar si una variable es igual a otra. Por ejemplo, en C/C++ si se pretende comparar estructuras o vectores, es necesario hacer un sobrecarga del operador igualdad (`operator==`) o definir la función correspondiente.

III. Configuración del Experimento

Para evaluar y validar las distintas combinaciones de técnicas mencionadas anteriormente, se ha utilizado como programa de pruebas la multiplicación de matrices de 16 elementos. Se han generado 10 versiones endurecidas del programa con las que se pretende evaluar y comprobar la idoneidad de nuestras estrategias de mitigación de fallos multi-hilo, así como el impacto de las estrategias de triplicación de datos en el rendimiento del programa.

Para ello, se ha propuesto endurecer el bucle más interno del programa de referencia mediante la replicación del flujo de instrucciones, el cual corresponde a la multiplicación y acumulación de cada uno de los resultados parciales de la matriz final. Además, se ha aplicado la protección de datos a algunas versiones mediante triplicación, protegiendo las matrices con los datos de entrada (multiplicando y multiplicador) y de verificación de resultados, que están alojadas en la sección de `rodata`, además de la matriz donde se guardan los cálculos realizados, que se encuentran alojada en la sección `bss`.

La primera versión es la que se utilizará como referencia, a la cual no se le ha aplicado ninguna técnica de triplicación de datos o flujo u optimización por parte del compilador. Las versiones con endurecimiento de flujo (DWR y TMR) y con triplicación de datos están denotadas por H2D2 y H3D3 respectivamente. De forma análoga, H2 y H3 corresponden a las versiones DWR y TMR sin triplicación de datos. Como se menciona en el apartado anterior, es opcional la sincronización de entrada en las versiones TMR, por ello se ha generado otra versión donde se ha eliminado el código de sincronización (H3_1 y H3D_1). Además, para completar el estudio, se han generado las versiones con el endurecimiento clásico añadiendo

la triplicación de datos (H3D_2T1 y H3D3_3T1) y sin triplicación (H1_T2 y H1_T3).

Para este experimento se ha usado el simulador *Simics* con precisión a nivel de instrucción. En él se ha simulado la placa *Versatile Express* caracterizada por ofrecer la posibilidad de poder utilizar distintos procesadores multinúcleo de la familia *Cortex-A9* de *ARM*. Así pues, el simulador se ha configurado atendiendo a la versión del programa con la que se trabaja. Cuando se evalúa el programa de referencia o las aproximaciones clásicas, con o sin triplicación de datos, caracterizadas por utilizar un solo núcleo, se utiliza la versión *ARM Cortex-A9Mx1*. Los DWR-R, con y sin triplicación de memoria, caracterizados por el uso de dos núcleos hacen uso del *Cortex-A9Mx2*. Finalmente, el procesador *Cortex-A9Mx4* se utiliza para comprobar las versiones TMR con y sin protección de memoria. Nótese que estas configuraciones son equivalentes a utilizar un procesador *Cortex-A9Mx4* en el que los núcleos sin actividad se encuentren en un bucle sin fin por defecto, pero por motivos de eficiencia del simulador se ha optado por la opción que más se ajusta a las características del programa.

La inyección esta dirigida por la herramienta *Fault Injection Manager* (FIM)[16]. FIM es un director de campañas que está a cargo de llamar a la plataforma que inserta un fallo, reiniciarla después de cada inyección y del etiquetado del resultado de la inyección del fallo. Los fallos son etiquetados como *unACE* cuando se realiza un inyección y el fallo no afecta al resultado de la salida del programa, *SDC* cuando el resultado no es correcto pero el programa acaba, y *HANG* si no acaba o supera un tiempo límite.

Para dotar al simulador *Simics* de capacidades de inyección de fallos se ha desarrollado un *plugin* al efecto, con la característica de que las inyecciones sean no intrusivas. Es decir, el simulador introduce un fallo en un bit aleatorio del recurso bajo evaluación (registros o secciones de memoria como `data`, `bss`, `rodata` y `stack`) sin hacer uso de ninguna rutina de interrupción.

El inyector se ha configurado para realizar 100 inyecciones por recurso. Por tanto, se han realizado 1800 inyecciones de fallos los registros de las versiones con un solo núcleo, 3600 en las versiones DWR-R multi-hilo y 5400 en los TMR multi-hilo. De forma análoga se han realizado 100 fallos por sección de memoria en las versiones no triplicadas y 300 en las versiones con los datos triplicados. Los programas han sido evaluados teniendo como referencia una ejecución sin fallos de los mismos y añadiendo un tiempo de recuperación de 1000 ciclos. En caso de superar esta restricción, el programa se consideran que no cumple con los requisitos validos y se etiqueta como *HANG*.

IV. Resultados

Las figuras 4 y 5 muestran los resultados obtenidos después de realizar las campañas de inyección de fallos a todas las versiones bajo evaluación (11 en nuestro caso). En ellas se puede observar la variación relativa en tanto por uno al programa de prueba sin endurecer, para las 5 técnicas de endurecimiento sin triplicación de memoria (H2, H3, H3_1, H1T2 y H1T3) y con triplicación de memoria (H3D2, H3D3, H3D_1, H3D_2T1, H3D_3T1) aplicadas a: todos los registros del banco de registros (`registros`), a la sección de datos de solo lectura (`rodata`), a la sección de datos inicializados (`bss`) y a la pila (`stack`) respectivamente.

En la figura 4 se puede observar cómo todas las técnicas de endurecimiento tienen un efecto positivo en los registros, con la mejora de unACE alrededor de 0.1×, la reducción de SDC en 0.8×. La tasa de HANG presenta variaciones distintas, por ejemplo las versiones H2 y H3 empeoran alrededor del 0.05×, mientras que las versiones H3_1, H1T2 y H1T3 mejoran alrededor del 0.2×. Observando las secciones de memoria `rodata`, `data` y `bss`, se puede apreciar que apenas ven alterado su comportamiento. Solo hay 3 casos en la sección de datos inicializados (`data`) donde los resultados presentan un notable cambio, H2 y H1T2 consiguen

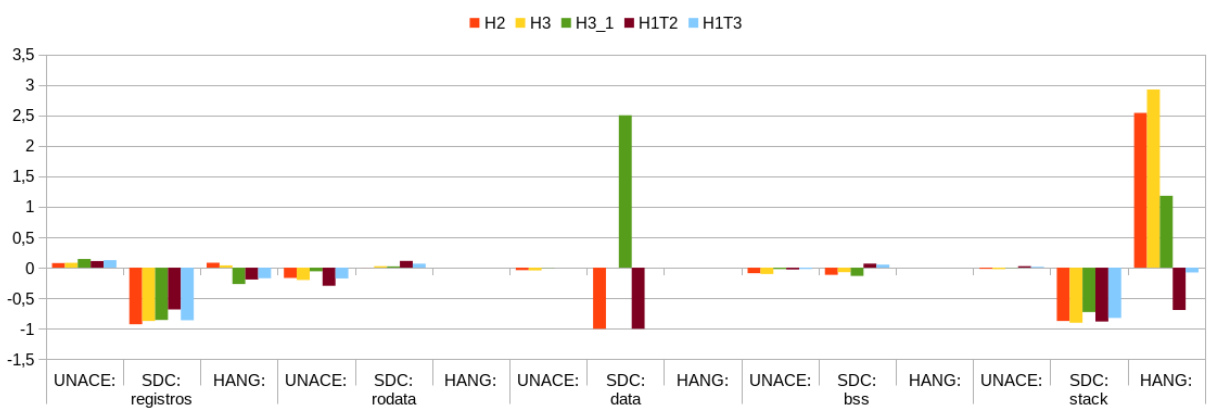


Fig. 4. Simulaciones para las versiones endurecidas con las técnicas multi-hilo y clásicas. La gráfica muestra las diferentes versiones de endurecimiento de flujos sin aplicar triplicación a los datos que residen en memoria. Los resultados están normalizados con la aplicación de referencia sin endurecimiento (línea base). Los resultados unACE positivos son mejores y los resultados SDC y HANG negativos son mejores.

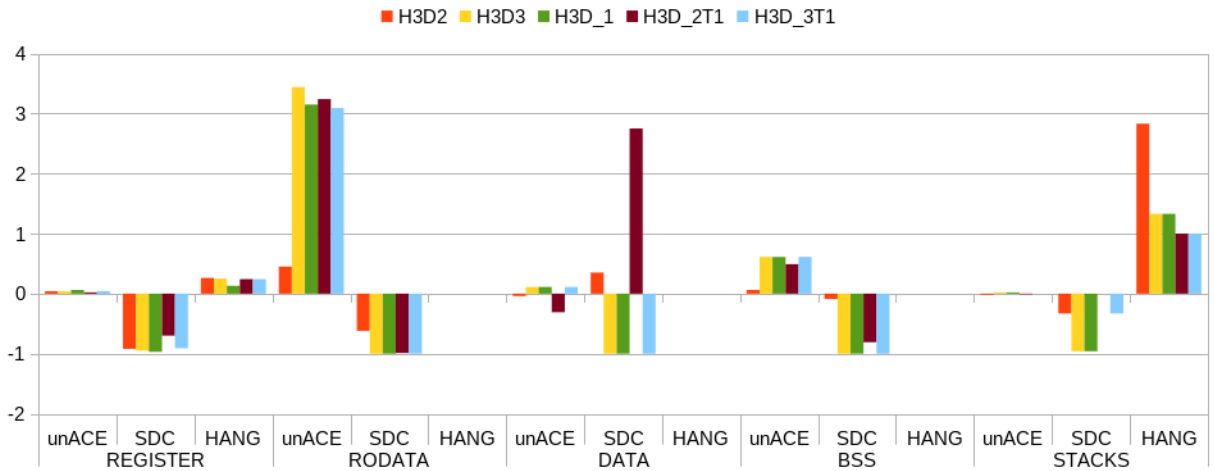


Fig. 5. Simulaciones para las versiones endurecidas con las técnicas multi-hilo y clásicas. La gráfica muestra las diferentes versiones de protección de flujo de datos aplicando triplicación a los datos que residen en memoria. Los resultados están normalizados con la aplicación de referencia sin endurecimiento (línea base). Los resultados unACE positivos son mejores y los resultados SDC y HANG negativos son mejores.

eliminar por completo los fallos SDC (-1×) y el caso de la técnica H3_1, que presenta una degradación de un factor 2.5× de los fallos tipo SDC. En el caso de los datos no inicializados (`bss`), las técnicas presentaron un comportamiento similar a la aplicación de referencia. Si se observa la tasa de SDC, las aplicaciones H2, H3 y H3_1 presentan una mejora alrededor del 0.1×, mientras que las H1T2 Y H2T3 empeoran en torno al 0.05×. La sección de `stack` presenta una variación importante entre los SDC y los HANG. Mientras que los SDC presentan una reducción significativa alrededor del 0.8×, los HANG presenta altas variaciones en las versiones H2, H3 y H3_1 (2.5×, 2.9× y 1.18× respectivamente), al contrario que sus versiones clásicas H1T2 y H1T3 (-0,7× y -0,08×), que mejoran y muy notablemente en el caso de H1T2. Como se esperaba, las técnicas de endurecimiento que modifican el flujo de trabajo ya sea en un solo núcleo o en varios, solo son capaces de proteger las variables activas durante el instante del programa, pero las variables con un tiempo de vida más largo que residen en la memoria no son alcanzadas con este tipo de protecciones, de ahí el reducido impacto de la misma en las secciones de memoria.

En la figura 5 se puede observar las mejoras introducidas cuando se ha triplicado la memoria de datos. Esta gráfica muestra la misma tendencia en los registros, donde no se aprecian grandes variaciones entre las versiones con datos triplicados y sin triplicar. En este caso, la gráfica si presenta importantes mejoras en las distintas secciones de memoria. Por ejemplo, la sección de `rodata` muestra claramente mejoras en todas las versiones cercanas a 3×, salvo H3D2 que presenta una mejora pero no supera el 1×. Además, se puede observar cómo la tasa de SDC se reduce por completo en la mayoría de ellas (-1×). La sección `data` solo presenta mejoras en los casos de triplicación del flujo de datos donde la recuperación de SDC es completa (-1×), en el caso de la duplicación la tasa de SDC aumenta, siendo más acusada en la versión clásica (2.75×). La sección de `bss` muestra mejoras en todos los casos estudiados, aunque los más llamativos son los que presentan menor tasa de mejora (H3D2 y H3D_2T1) que corresponde en ambos casos a la versión DWR-R (-0.1× y 0.8×), el resto de versiones (TMR) consiguen eliminar por completo los SDC (-1×). Finalmente, la sección `stack` presenta una traslado de fallos de SDC a HANG, siguiendo la tendencia presentada en el caso sin triplicación de datos. En este caso, la triplicación de datos muestra mayores efectos positivos que las versiones donde no se ha triplicado los datos. Es más, el efecto más relevante lo encontramos en las recuperaciones de la sección de memoria `rodata`. Debido a la política del mapa de memoria del dispositivo y, como el propio nombre de la sección indica, son variables de solo lectura y no se pueden modificar para corregir el fallo detectado. Aun así, presentan una tasa de recuperación presentada muy elevada. Otro efecto positivo es que las secciones de memoria no ven incrementada su tasa de HANG en ningún momento, y las reducciones en los SDC redundan en un aumento la tasa de unACE.

La Figura 6 presenta la métrica *Mean Work to Failure* (MWTF)[17], que captura el compromiso (*trade-off*) entre la cobertura frente a fallos y el rendimiento de una aplicación. En ella se observan los sobrecostes de aplicar técnicas de duplicación o triplicación, aún cuando éstas presentan resultados positivos. En el caso de las versiones sin triplicación de datos, las mejoras obtenidas tras la aplicación de redundancia en el flujo no son suficientes para compensar los sobrecostes en rendimiento, pero si se observan las versiones con triplicación de datos y de flujos, la aplicación de las técnicas de endurecimiento empieza a ser positiva. Aún más, las versiones con mejores tasas son las versiones paralelizadas, ya que consiguen disminuir

la tasas de errores sin incurrir en graves sobrecostes temporales como se observan en las versiones clásicas.

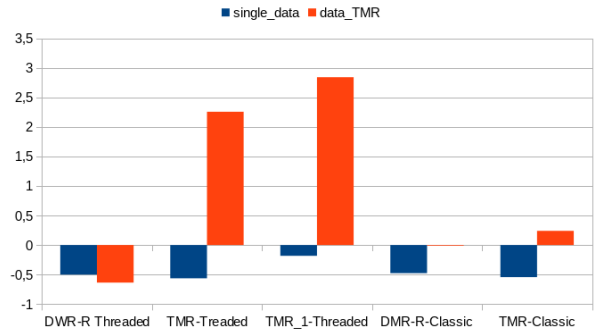


Fig. 6. Mean Work to Failure de las aplicaciones bajo evaluación. Todos los programas están normalizados con el programa de referencia (linea base).

V. Conclusiones

Este artículo presenta una técnica software para la protección de datos frente a fallos inducidos por radiación basada en una estrategia multi-hilo con triplicación de datos en memoria. La técnica propuesta extiende las técnicas de duplicación y triplicación de flujos propuestas por los mismo autores, teniéndose ahora en cuenta la problemática de la persistencia de los datos en memoria, donde la presencia de fallos corrompe los datos sin posibilidad de recuperación. Los resultados obtenidos muestran que a pesar de ser consideradas técnicas que implican un sobrecoste evidente en el número de instrucciones y memoria empleada, la paralelización de las técnicas de triplicación de flujos obtiene mejores resultados que las técnicas duplicación, llegándose incluso a mejorar los tiempos de recuperación frente a errores.

Agradecimientos

Este trabajo ha sido financiado por el Ministerio de Economía y Competitividad de España y el Fondo Europeo de Desarrollo Regional (FEDER) mediante el proyecto de investigación "*Evaluación temprana de los efectos de radiación mediante simulación y virtualización. Estrategias de mitigación en arquitecturas de microprocesadores avanzados*" (Ref: ESP2015-68245-C4-3-P MINECO/FEDER, UE).

Referencias

[1] Robert C. Baumann, "Radiation-induced soft errors in advanced semiconductor technologies," *IEEE Transactions on Device and Materials Reliability*, vol. 5, no. 3, pp. 305–315, 9 2005.

[2] X. Iturbe, B. Venu, E. Ozer, and S. Das, "A triple core lock-step (tcls) arm® cortex®-r5 processor for safety-critical and ultra-reliable applications," in *2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshop (DSN-W)*, June 2016, pp. 246–249.

[3] H. Quinn, Z. Baker, T. Fairbanks, J. L. Tripp, and G. Duran, "Software resilience and the effectiveness of software mitigation in microcontrollers," *IEEE Transactions on Nuclear Science*, vol. 62, no. 6, pp. 2532–2538, Dec 2015.

[4] A. Serrano-Cases, Y. Morilla, P. Martín-Holgado, S. Cuenca-Asensi, and A. Martínez-Álvarez, "Non-intrusive automatic compiler-guided reliability improvement of embedded applications under proton irradiation," *IEEE Transactions on Nuclear Science*, pp. 1–10, 2019.

[5] A. Martinez-Alvarez, S. Cuenca-Asensi, F. Restrepo-Calle, F. R. Palomo Pinto, H. Guzman-Miranda, and M. A. Aguirre, "Compiler-directed soft error mitigation for embedded systems," *IEEE Transactions on Dependable and Secure Computing*, vol. 9, no. 2, pp. 159–172, March 2012.

[6] S. Cuenca-Asensi, A. Martinez-Alvarez, F. Restrepo-Calle, F. R. Palomo, H. Guzman-Miranda, and M. A. Aguirre, "A novel co-design approach for soft errors mitigation in embedded systems," *IEEE Transactions on Nuclear Science*, vol. 58, no. 3, pp. 1059–1065, June 2011.

[7] G. S. Rodrigues, F. Rosa, F. L. Kastensmidt, R. Reis, and L. Ost, "Investigating parallel tmr approaches and thread disposability in linux," in *2017 24th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, Dec 2017, pp. 393–396.

[8] G. S. Rodrigues, F. Rosa, Á. B. de Oliveira, F. L. Kastensmidt, L. Ost, and R. Reis, "Analyzing the impact of fault-tolerance methods in arm processors under soft errors running linux and parallelization apis," *IEEE Transactions on Nuclear Science*, vol. 64, no. 8, pp. 2196–2203, Aug 2017.

[9] G. S. Rodrigues, F. L. Kastensmidt, R. Reis, F. Rosa, and L. Ost, "Analyzing the impact of using pthreads versus openmp under fault injection in arm cortex-a9 dual-core," in *2016 16th European Conference on Radiation and Its Effects on Components and Systems (RADECS)*, Sept 2016, pp. 1–6.

[10] Saurabh Hukerikar, Keita Teranishi, Pedro C. Diniz, and Robert F. Lucas, "Redthreads: An interface for application-level fault detection/correction through adaptive redundant multithreading," *International Journal of Parallel Programming*, vol. 46, no. 2, pp. 225–251, Apr 2018.

[11] J. S. Monson, M. Wirthlin, and B. Hutchings, "Fault injection results of linux operating on an fpga embedded platform," in *2010 International Conference on Reconfigurable Computing and FPGAs*, Dec 2010, pp. 37–42.

[12] H. So, M. Didehban, A. Shrivastava, and K. Lee, "A software-level redundant multithreading for soft/hard error detection and recovery," in *2019 Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2019, pp. 1559–1562.

[13] A. Serrano-Cases, S. Cuenca-Asensi, and A. Martínez-Álvarez, "Protección de software frente a radiación en procesadores multi-núcleo sin sistema operativo," in *2018 Jornadas Sarteco*. Sept. 2018, Zenodo.

[14] A. Serrano-Cases, F. Restrepo-Calle, S. Cuenca-Asensi, and A. Martínez-Álvarez, "Softerror mitigation for multi-core processors based on thread replication," in *2019 IEEE Latin American Test Symposium (LATS)*, March 2019, pp. 1–5.

[15] Steven K. Reinhardt and Shubhendu S. Mukherjee, "Transient fault detection via simultaneous multithreading," *SIGARCH Comput. Archit. News*, vol. 28, no. 2, pp. 25–36, May 2000.

[16] J. Isaza-González, A. Serrano-Cases, F. Restrepo-Calle, S. Cuenca-Asensi, and A. Martínez-Álvarez, "Dependability evaluation of cots microprocessors via on-chip debugging facilities," in *2016 17th Latin-American Test Symposium (LATS)*, April 2016, pp. 27–32.

[17] George A. Reis, Jonathan Chang, Neil Vachharajani, Ram Rangan, David I. August, and Shubhendu S. Mukherjee, "Design and evaluation of hybrid fault-detection systems," in *Proceedings - International Symposium on Computer Architecture*. 2005, pp. 148–159, IEEE.

[18] Shubhendu S. Mukherjee, Christopher Weaver, Joel Emer, Steven K. Reinhardt, and Todd Austin, "A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor," in *Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture*, Washington, DC, USA, 2003, MICRO 36, pp. 29–, IEEE Computer Society.

Ordenamiento de canales del sensor GSENSE400 en modo STD para el instrumento IMaX+

Eduardo Magdaleno[1], Manuel Rodríguez[1], David Hernández[2], María Balaguer[2], D. Orozco Suárez[2], Daniel Álvarez[2], A. López Jiménez[2], José Carlos del Toro Iniesta[2], Basilio Ruiz Cobo[3]

Resumen—**GSENSE400 en un sensor de imagen CMOS 2048x2048 que suministra los datos a través de 8 canales diferenciales. Esos datos han tenido que ser ordenados en streaming con el fin de que el instrumento IMaX+ transmita imágenes en el formato filas x columnas deseado. Este trabajo describe el diseño e implementación del módulo swapping que realiza esta tarea.**

Palabras clave—**IMaX+, Sunrise3, firmware, FPGA.**

I. INTRODUCCIÓN

SUNRISE III es la tercera de las misiones en la que se va a lanzar, en un globo y desde el círculo polar ártico, un telescopio a la estratosfera con el fin de observar durante unos días de manera ininterrumpida el Sol [1].

El mayor cambio de Sunrise III respecto a las misiones anteriores es el conjunto de instrumentos, ya que el telescopio y la góndola serán similares [1].

En concreto, se está construyendo un nuevo magnetógrafo, denominado IMaX+, que proporciona imágenes del Sol de 50x50 arcmin2 en varias bandas estrechas (~8pm de anchura), de la que se puede extraer la información cuatro estados de polarización [1]. En este instrumento, respecto al antiguo IMaX, se ha rediseñado completamente la electrónica, empleando componentes tecnológicos actuales que redundarán también en un sistema mucho más pequeño, dejando más espacio para los instrumentos a bordo.

En lo que respecta a la cámara del sensor IMaX+, se ha desarrollado un hardware que incluye dos PCBs, el primero con el sensor CMOS, y el segundo, con una FPGA Artix-7 XC7A50T-2CSG325C encargada de controlar el sensor y de comunicarse con la DPU del instrumento a través de una interfaz CoaxPress [2-3]. En la figura 1 se muestra una foto del primer prototipo implementado de la cámara de IMaX+. En posición vertical está situada la PCB del sensor y en posición horizontal, la de la FPGA.

El vuelo de la misión Sunrise 3 está previsto para el verano septentrional de 2020.

El resto del presente trabajo está organizado como sigue: en la segunda sección se describen las principales características de los dispositivos más relevantes.

[1]Dpto. de Ingeniería Industrial, Univ. de La Laguna, e-mail: emagcas@ull.edu.es
[2]Grupo de Física Solar, Inst. de Astrofísica de Andalucía (IAA-CSIC), e-mail: dhdez@iaa.es
[3]Grupo de Física Solar, Inst. de Astrofísica de Canarias (IAC), e-mail: brc@iac.es

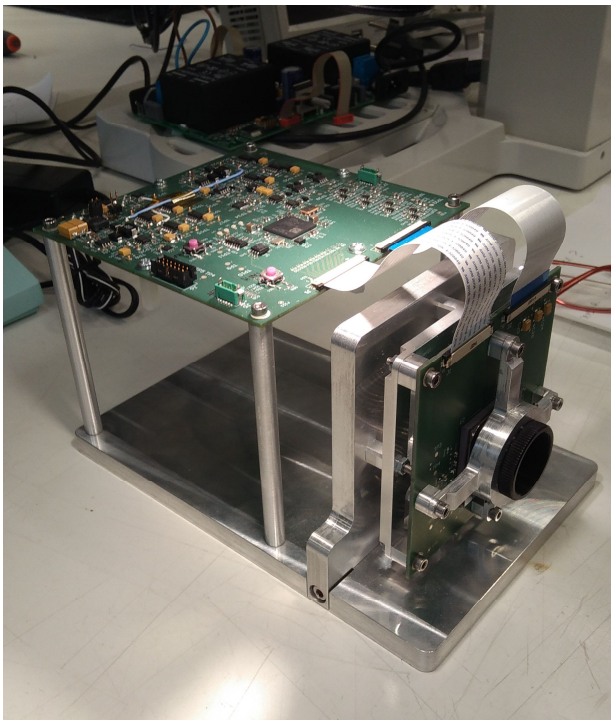


Fig. 1. Imagen del primer prototipo de la cámara del instrumento IMaX+.

En la sección tercera se describe en detalle en módulo desarrollado para ordenar la imagen que suministra el sensor. En la cuarta se presentan los resultados y, por último, se finaliza con las conclusiones.

II. DESCRIPCIÓN DEL SENSOR DE IMAGEN

En la fase de diseño del prototipo optó por separar en dos tarjetas los elementos más relevantes. Por un lado, el sensor GSENSE400 de GPIXEL y, por otro lado, la FPGA Artix-7 XC7A50T-2CSG325C que configura y se comunica con el sensor, presentando también una interfaz CoaXPress para la comunicación con el host o DPU del sistema [4].

A. Características principales del sensor GSENSE400

El sensor que se ha escogido para incorporarlo al sistema de detección tiene una serie de características que cumplen con los requisitos exigidos por las comisiones científicas del proyecto.

El sensor GSENSE400 es un sensor de imagen CMOS de resolución de 4 megapíxeles (2048x2048) con píxeles de fotodiodo de 11µm. El sensor presenta un ruido de

lectura extremadamente bajo de 1.47e-. Tiene dos modos de operación, el modo STD en 48 fotogramas por segundo o el modo HDR que está optimizado para aplicaciones de alto rango dinámico con 24 fotogramas por segundo.

La característica de alta sensibilidad, bajo nivel de ruido de lectura y alto rango dinámico lo hace perfecto para una variedad de aplicaciones científicas, como la que nos ocupa.

El sensor dispone de un conversor AD de 12 bits, un sensor de temperatura, un PLL y una interfaz SPI para labores de control. Para configurar el detector y ponerlo en marcha adecuadamente, hay que escribir en unos bancos de memoria del mismo a través de esta interfaz [4].

Para este proyecto, se ha optado por el modo STD que se explicará a continuación.

B. Descripción del formato de salida de la imagen

El sensor dispone de 8 canales diferenciales que proporcionan los datos de los píxeles de 12 bits en formato serie a 300MHz. Las filas del sensor CMOS se leen o se resetean en ranuras temporales 513 ciclos de reloj de píxel (25MHz) a través de unas señales de control que debe enviar la FPGA Artix-7.

En el modo HDR, una ranura temporal consta de 2 fases: fase de lectura de una fila y fase de reset de una fila. Con este modo se obtiene, para cada dirección, dos lecturas, una en alta ganancia y otro en baja ganancia. Este modo no está configurado.

En el modo STD, una ranura temporal consta de 4 fases: fase de lectura de una fila, fase de reset de una fila, y fases de lectura y reset de las filas siguientes, respectivamente. En la figura 2 se muestra cómo se leen las filas M y siguientes y se resetean las filas N y siguientes en dos ranuras temporales.

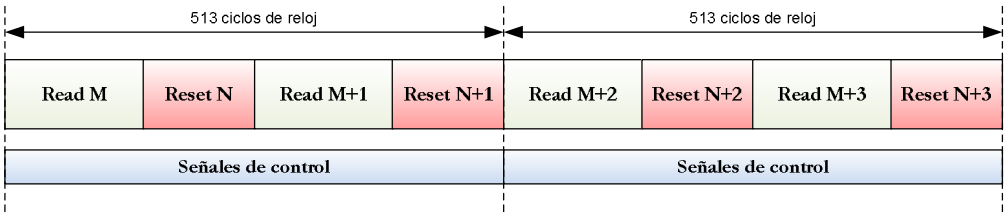


Fig. 2. Temporización de la lectura y reseteo de las filas de la imagen del sensor en modo STD.

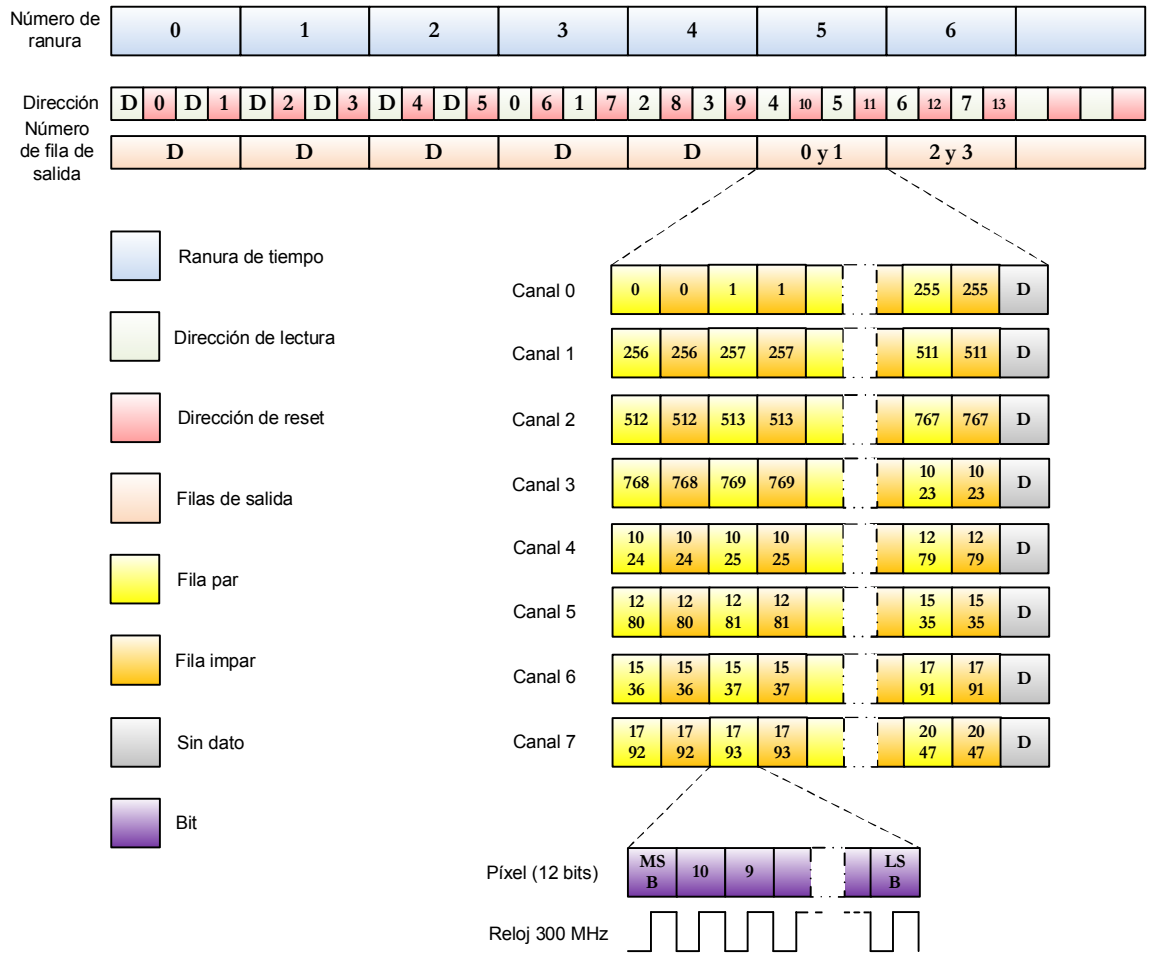


Fig. 3. Formato de salida de datos con el sensor en modo STD (estándar).

Este modo de reseteo y lectura se realiza de manera simultánea para los 8 canales, que se reparten las 2048 filas del sensor, según se aprecia en la figura 3, donde en la que se hace la distinción entre filas pares e impares, de tal manera que, en cada ranura temporal una fila para y otra impar son leídas/reseteadas en una estructura de pipeline. Puede apreciarse que, desde la orden de lectura hasta que el sensor proporciona los datos, pasa el tiempo correspondiente a dos ranuras, debido a la estructura pipeline de salida del sensor.

Cada fila tiene 2048 píxeles. Entonces, y según se aprecia en la figura 3, cada dos filas simultáneamente (par e impar), el sensor suministra los datos así: el canal cero, los píxeles del 0 al 255 de esas dos filas; el canal 1, los píxeles del 256 al 511, etcétera.

Así pues, hay que ordenar los datos en dos niveles: por paridad de filas y por canales.

III. DESCRIPCIÓN DEL MÓDULO DE ORDENACIÓN IMPLEMENTADO

Como ya se ha comentado, la FPGA Artix-7 se encarga de la configuración y la comunicación con el sensor. El firmware implementado consta fundamentalmente de tres módulos según se aprecia en la figura 4. El módulo de clocking/reset genera todos los relojes necesarios para el funcionamiento del firmware, así como un reset global. Por otra parte, se ha implementado, con un IP core comercial, una interfaz CoaxPress con la que se comunica con el host o DPU (tanto datos de imagen como valores de configuración y otros comandos).

El tercer módulo implementado es el driver del sensor. Como puede apreciarse, el driver envía, a través de un bus SPI, los valores de configuración del sensor, que también pueden leerse; y también envía las señales de control para gestionar las órdenes de lectura y reset de las filas de las imágenes por él obtenidas. El driver recibe los datos a través de los 8 canales diferenciales, a 300MHz. Cada canal de entrada conectada de manera diferencial a la FPGA consta de un módulo que pasa las señales de un formato unipolar y un deserializador de 1 a 12, que agrupa los datos serie en píxeles de 12 bits, pasando de 300 a 25MHz como reloj de manejo de datos.

Para realizar el ordenamiento de los datos que provienen simultáneamente a través de los 8 canales de la manera descrita en el apartado II, se ha implementado el módulo swapping, que puede apreciarse en la figura 5.

Los datos ya deserializados procedentes de cada canal del sensor son introducidos en unos submódulos que se han denominado channel_swap. Nótese que dentro de estos bloques los píxeles (de 12 bits), se agrupan de 8 en 8 (96 bits). Esto se debe a que los datos, una vez ordenados, se van a enviar a través de la interfaz CoaXPress empleando ese tamaño de datos.

Cada uno de los módulos channel_swap se compone de dos shift-register de profundidad 7 y dos FIFOs de 96bits y 512 de profundidad, como se aprecia en la figura 6.

Así pues, este módulo recibe los datos. Con las señales ce_even y ce_odd, en cada canal se distribuyen los píxeles entre pares e impares y, tras 16 ciclos de reloj, se activan las señales load_even y load_odd, para agrupar los datos en paquetes pares e impares de 8 píxeles (96 bits). Estos paquetes, se almacenan en las FIFOs (par e impar) con las señales wr_fifo_even y wr_fifo_odd. El control en la lectura de estas FIFOs hará que los datos de la imagen queden ordenados por filas en lugar de por canal y paridad.

Todas las señales anteriormente nombradas son gestionadas por una máquina de estados, que también controla un contador que es el encargado de generar la señal de selección (con_rd) de un multiplexor 16 a 1 que selecciona adecuadamente el dato disponible a la salida de cada FIFO.

En la figura 7 se muestra una simulación en la que se aprecia cómo las señales ce_even y ce_odd se van conmutando, realizando un demultiplexor 1 a 2, y distribuyendo así los datos de filas pares e impares entre los dos shift-register para cada canal. Puede apreciarse también que, después de 16 ciclos, se levantan las señales load_even y load_odd, mostrando el dato de entrada a las FIFOs (señales fifo_input marcadas en la figura).

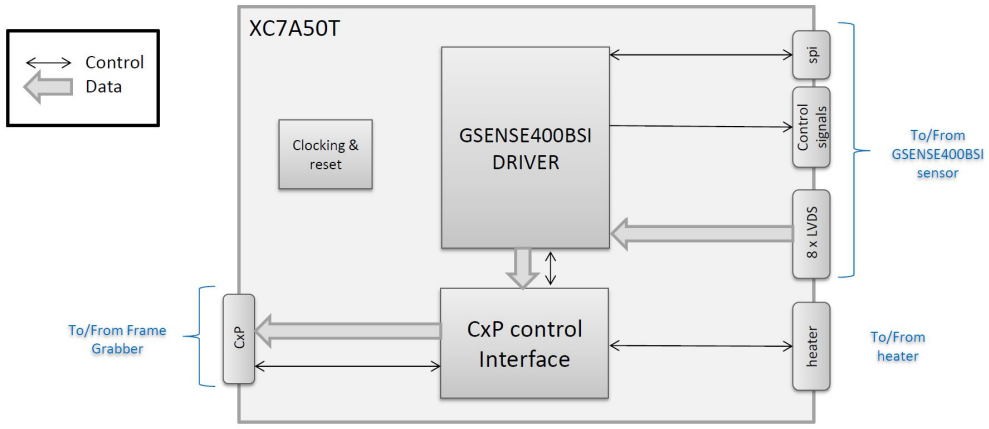


Fig. 4. Diagrama de bloques del firmware implementado en la FPGA Artix-7.

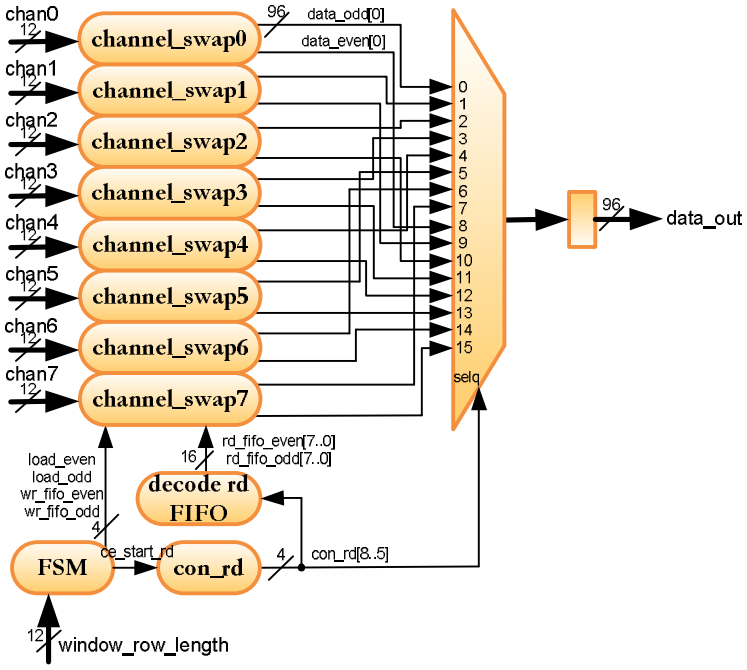


Fig. 5. Diagrama de bloques del módulo de ordenamiento (swapping).

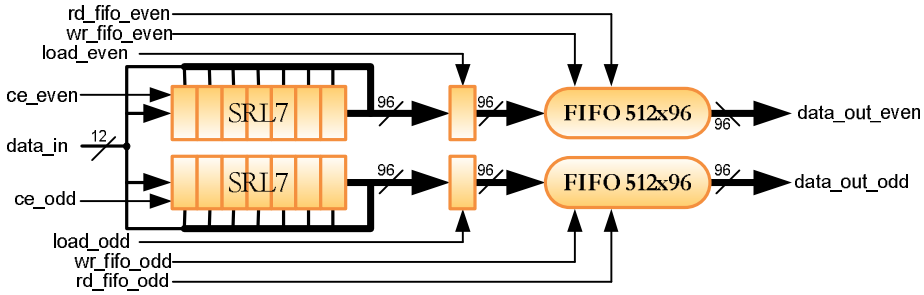


Fig. 6. Diagrama de bloques del módulo channel_swap.

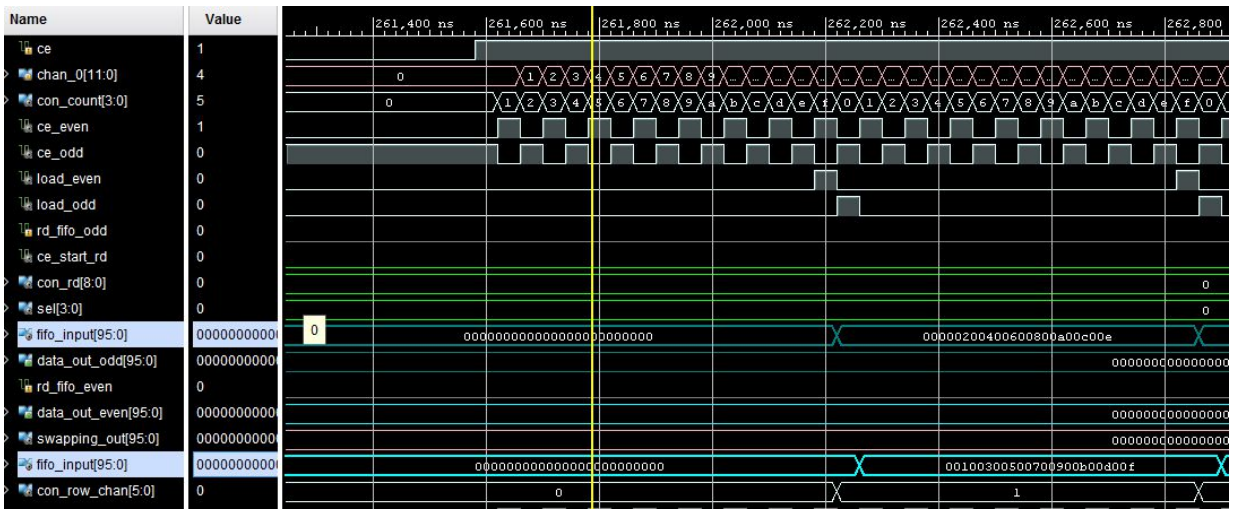


Fig. 7. Cronograma de simulación de la entrada de datos al módulo de ordenamiento.

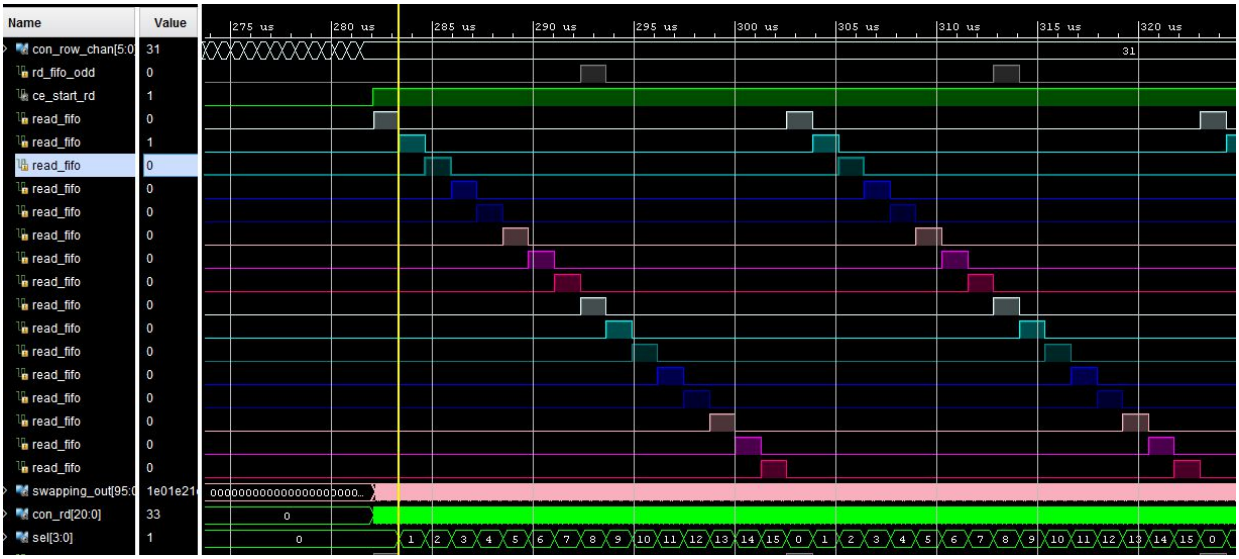


Fig. 8. Cronograma de simulación de la lectura de las 16 FIFOs implementadas.

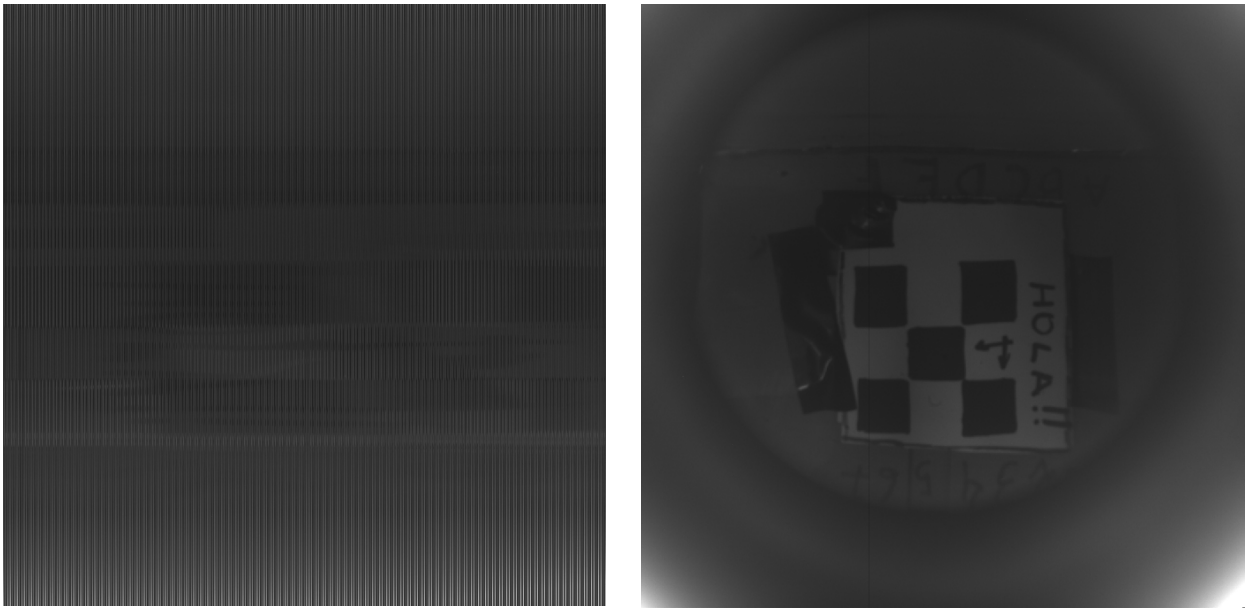


Fig. 9. Imagen de prueba tomada con el prototipo de la cámara IMaX+ (izquierda canales desordenados, derecha canales ordenados).

Fig. 10. Imagen de prueba tomada por el prototipo IMaX+

En la figura 8 se aprecia lectura de las 16 FIFOs, multiplexadas en el tiempo y en streamming. Las señales con_rd y sel (en verde), provenientes de la máquina de estado (FSM), gestiona la lectura. empezando a leer los datos de la FIFO par canal 0, FIFO par canal 1, … FIFO par canal 7, FIFO impar canal 0, FIFO impar canal 1, … FIFO impar canal 7 y vuelta a empezar hasta que se recorran todas filas de la imagen del sensor seleccionadas (señal window_row_length en la figura 5). La swapping_out (en rosa), recibe los datos agrupados (en 8 píxeles) ya ordenados.

IV. RESULTADOS Y CONCLUSIONES

Se ha implementado en VHDL un módulo de ordenamiento de canales de datos del sensor GSENSE400 de GPIXEL.

Este módulo se ha integrado satisfactoriamente dentro del driver sensor y ya se encuentra operativo. En la figura 9 se muestra una imagen de prueba obtenida por

el sensor. A la izquierda se aprecia la imagen sin emplear el módulo de ordenamiento y, a la derecha, con el módulo implementado ya integrado en el sistema. Las imágenes se han enviado a través de la interfaz CoaXPress y capturadas usando un frame grabber comercial [3]. En la figura 10 se muestra otra captura ordenada con el módulo de swapping.

La realización de la tarea de ordenamiento en modo local en el propio driver del sensor implementado en la FPGA Artix-7 permite liberar de esta tarea a la DPU o host del sistema.

En lo que respecta a los recursos de la FPGA Artix-7 XC7A50T-2CSG325C, se muestran en la tabla siguiente.

TABLA I

RECURSOS DE LA FPGA EMPLEADOS EN EL REORDENAMIENTO.

Recursos	Uso	Disponibilidad	%
LUT	1088	32600	3.34
FF	1164	65200	1.79
BRAM	24	75	32.00

Claramente, el recurso más comprometido es el de memoria (BRAM), empleado para configurar las 16 FIFOs necesarias para realizar la operación (32%). No obstante, el resto de los recursos empleados por el driver del sensor se emplea en labores de control y configuración y apenas se emplea mucha memoria adicional.

AGRADECIMIENTOS

El presente trabajo ha sido financiado mediante el proyecto ESP2016-77548-C5-2-R del Ministerio de Economía, Industria y Competitividad.

REFERENCIAS

[1] B. Ruiz, Física solar espacial: PHI para Solar Orbiter e IMaX y SP para Sunrise. Memoria científico-técnica de proyectos coordinados. Convocatoria 2016 de Proyectos de Excelencia y Proyecto Retos. Dirección General de Investigación Científica y Técnica. 2016.

[2] Xilinx, 7 Series FPGAs Data Sheet: Overview. Product Specification. DS180 (v2.5). Xilinx. 2017. Disponible en: https://www.xilinx.com/support/documentation/data_sheets/ds180_7Series_Overview.pdf (accedido el 28 de septiembre de 2017).

[3] EASii IC. CoaxPress Device IP Specification.

[4] Gpixel, 4 Megapixels Scientific CMOS Image Sensor. Datasheet V1.5

Conectividad de sistemas

Protocolo de coordinación de enjambres de VANTs para misiones planificadas

Francisco Fabra, Pablo Reyes, Carlos T. Calafate, Juan Carlos Cano, Pietro Manzoni[1] y Willian Zamora [2]

Resumen— **Los Vehículos Aéreos No Tripulados (VANTs), comúnmente conocidos como drones, son una herramienta cada vez más usada en situaciones críticas, especialmente cuando el acceso al destino es difícil o peligroso. En estas situaciones, un enjambre de drones puede proporcionar beneficios adicionales, como cubrir rápidamente áreas extensas acortando el tiempo de misión, o elevando cargas excesivas para un solo dron. En este trabajo se presenta MUSCOP, un protocolo que sincroniza el vuelo de un conjunto de drones del tipo multirrotor mientras siguen una misión planificada. Los resultados obtenidos muestran que MUSCOP proporciona al enjambre un grado de cohesión elevado, incluso cuando hay pérdida de mensajes en el canal de comunicación, con unos retardos de sincronización y errores de posicionamiento muy reducidos en comparación con un escenario ideal.**

Palabras clave— **VANT; enjambre; coordinación de vuelo; ArduSim.**

I. Introducción

HOY en día, los Vehículos Aéreos No Tripulados (VANTs), más conocidos como drones, se están utilizando para una gran cantidad de aplicaciones, algunas de ellas ni siquiera imaginadas hace una década. Algunas de esas soluciones requieren, sin embargo, más de un dron para ser eficientes, o pueden ser optimizadas utilizando un enjambre de drones. Por ejemplo, varios drones pueden transportar diferentes sensores para captar un conjunto de datos más completo, o pueden llevar el mismo tipo de sensor para cubrir un área mayor en menos tiempo. Ejemplos de tales aplicaciones pueden ser la búsqueda de plagas en la agricultura a gran escala [1, 2], la grabación de fauna salvaje [3], operaciones de búsqueda y rescate [4, 5], o la vigilancia de fronteras [6], entre otras.

Aunque en algunos casos se puede desplegar un grupo de drones siguiendo misiones independientes, sin que la cohesión del enjambre sea un requerimiento (por ejemplo en espectáculos aéreos), hay otros casos en los que es necesario definir una única misión para todo el enjambre, garantizando que todos los drones mantienen la formación. El mayor reto en este tipo de soluciones es cómo garantizar la perfecta sincronización durante todo el vuelo, pues la comunicación entre ellos se realiza mediante un canal con pérdida de mensajes, y, además, cuando se emplea un enjambre de grandes dimensiones puede haber retrasos significativos en la misión si se compara con el vuelo de un solo dron.

Para alcanzar estos objetivos, en este trabajo se propone MUSCOP (Mission-based UAV Swarm Coordination Protocol), un innovador protocolo que coordina el vuelo de un enjambre de drones del tipo multicóptero para mantener estable la formación mientras se sigue una misión planificada. Este protocolo se adapta a cualquier formación deseada y escala adecuadamente a un número elevado de drones, incluso en presencia de pérdida de mensajes en el canal de comunicación, con retrasos en la sincronización muy reducidos. Todo esto se consigue utilizando cada waypoint o punto de paso de la misión como un punto de sincronización, permitiendo continuar con la misma únicamente cuando todos los drones han alcanzado la posición designada.

Se ha utilizado la plataforma de simulación ArduSim [7] para realizar experimentos realistas, con los que se ha comprobado que MUSCOP es capaz de mantener la formación de vuelo con errores de posicionamiento menores a 2 metros, valor obtenido para una velocidad de vuelo de *10 m/s*, y con un retraso temporal menor a 1 segundo, evitando colisiones en el enjambre. También se ha verificado que la complejidad de la misión es el factor que más afecta al tiempo de misión, siendo la sobrecarga introducida por la sincronización del protocolo prácticamente despreciable.

El resto del artículo se organiza como sigue: en la sección II se presentan diversos trabajos relacionados. En la sección III se detalla el protocolo propuesto, incluyendo la máquina de estados finitos que permite coordinar el enjambre, así como el formato de los mensajes enviados. La sección IV describe las misiones utilizadas en los experimentos y las métricas empleadas para evaluar el funcionamiento de MUSCOP. A continuación, en la sección V se incluye una validación detallada del protocolo. Finalmente, en la sección VI se presentan las conclusiones y trabajos futuros.

II. Trabajos relacionados

El uso de enjambres de drones es un requisito fundamental en diversas aplicaciones. Esta estrategia implica, sin embargo, resolver muchos retos tecnológicos, incluyendo la coordinación del enjambre, las comunicaciones y el control de cada dron. A continuación se incluye una visión general de trabajos relevantes en la literatura, en el contexto de los enjambres de drones.

En [8] se propone un sistema de control automático para enjambres de drones. Los autores formulan e

[1]Departamento de Ingeniería de Sistemas y Computadores, Universitat Politècnica de València (UPV), e-mail: frafabco@cam.upv.es, gigavuze@gmail.com, {calafate, jucano, pmanzoni}@disca.upv.es

[2]Facultad de Ciencias Informáticas, Universidad Laica Eloy Alfaro de Manabí, Manta, Ecuador, e-mail: willian.zamora@live.uleam.edu.ec

implementan un sistema de control para evitar colisiones entre los drones, manteniendo la cohesión de la formación. La solución se comporta bien con pérdida de mensajes, y escala correctamente en enjambres grandes.

En [9] se presenta un protocolo de comunicación para enjambres autónomos realizando misiones de búsqueda. Esta propuesta combina la comunicación entre drones con enrutamiento geográfico de mensajes para mejorar la eficiencia de la búsqueda. Por otro lado, en [10] se propone utilizar enjambres de drones para montar una infraestructura de comunicación inalámbrica en áreas que han sufrido un desastre. Para ello, se instalan agentes autónomos en cada dron para controlarlos de forma cooperativa.

En [11] se presenta una topología basada en redes Ad-hoc para controlar la movilidad de enjambres de drones. Sus principales características estudiadas se centran en la conectividad y en el área cubierta. De igual modo, en [12] se incluye un estudio exhaustivo de Flying Ad-hoc Networks (FANETs), donde se describen los principales problemas a resolver para desplegar redes Ad-hoc en drones. Los autores describen propiedades importantes como los cambios de topología, el modelo de propagación de la señal de comunicación, adaptabilidad, escalabilidad, latencia y ancho de banda.

En [13] se diseña un controlador de la formación del enjambre basado en una estructura virtual, que es ampliamente evaluado mediante simulación.

Este trabajo difiere de todos los anteriores ya que se propone un protocolo que establece y mantiene una formación de drones en tiempo real, mientras siguen una misión planificada. El líder del enjambre utiliza los waypoints de la misión como puntos de control para garantizar que el resto de drones mantienen sus posiciones relativas en el enjambre.

III. Protocolo MUSCOP

En esta sección se presenta el protocolo MUSCOP, que ha sido desarrollado para coordinar el vuelo de un enjambre de drones mientras siguen una misión previamente planificada. Para ello se ha utilizado ArduSim [7], una plataforma de simulación realista y precisa, que además permite desplegar fácilmente el código implementado en drones reales.

A. Visión general

MUSCOP ha sido diseñado en base al modelo maestro-esclavo, siendo el dron maestro el que sincroniza a los esclavos cada vez que llegan a un waypoint de la misión. Es más, antes de iniciar la misión, los esclavos reciben del maestro una misión copia de la misión del maestro, pero modificada para tener en cuenta su posición relativa al maestro dentro del enjambre, tal y como se detalla en la sección III-B. De este modo, cada dron se mueve de un waypoint al siguiente según su propia misión cada vez que el dron maestro envía la orden correspondiente. La formación de vuelo se mantiene estable durante toda la misión al utilizar los waypoints como puntos de sincronización. Entre waypoints, los drones mantienen prácticamente la misma posición relativa por tener programada la misma velocidad de vuelo para toda la misión.

Los mensajes utilizados para sincronizar el enjambre se transmiten del maestro a los esclavos y viceversa, siendo necesarios dos hilos de ejecución en cada dron para gestionar los mensajes (*TalkerThread* y *ListenerThread*). El hilo *TalkerThread* envía los mensajes a otros drones, mientras que el hilo *ListenerThread* los recibe, implementa la máquina de estados finitos del protocolo, y envía las órdenes adecuadas a la controladora de vuelo del dron en cuanto recibe los mensajes pertinentes.

B. Máquina de estados finitos

La figura 1 muestra la máquina de estados finitos que rige el comportamiento de los drones maestro y esclavos. Los círculos representan los estados, las flechas curvadas los mensajes enviados y recibidos, y las líneas rectas las transiciones entre estados. Las flechas curvadas a la izquierda de los estados indican los mensajes enviados (*TalkerThread*), mientras que las del lado derecho representan los mensajes recibidos (*ListenerThread*).

Las letras "M" y "S" hacen referencia al dron maestro y a los esclavos, respectivamente. Antes que los drones despeguen, el dron que se ubicará en el centro de la formación de vuelo ("C") se convierte en el maestro, y el resto de drones toman el rol de esclavos ("NC"). Al ceder el control del enjambre al dron situado en el centro se optimizan las comunicaciones, ya que los mensajes se transmiten del maestro a los esclavos y vice-versa, y se necesita minimizar la distancia entre transmisor y receptor para mitigar la pérdida de mensajes durante la transmisión.

Todos los drones comienzan en el estado *Start*. Los esclavos envían al maestro el mensaje *hello* para informar de su presencia, permitiéndole determinar el número de drones que formarán el enjambre. Cuando el usuario observa que el maestro ha identificado a todos los drones, utiliza el panel de control para cambiar al estado *Setup*. Entonces, el dron maestro decide cuál es la posición de cada esclavo en la formación de vuelo, y calcula la misión de cada esclavo teniendo en cuenta su posición relativa en la formación respecto al maestro. A continuación, envía el mensaje *data* a los esclavos con su respectiva misión. Seguidamente, los esclavos informan al maestro mediante el mensaje *dataAck* que han recibido la misión. Sólo cuando el maestro comprueba que todos los esclavos tienen la misión cambia éste al estado *Ready to fly*, enviando el mensaje *readyToFly* para que los esclavos también cambien al mismo estado, y envíen la correspondiente confirmación *(readyToFlyAck)*. Llegados a este punto, el dron maestro inicia el despegue (estado *Taking off*) y deja de enviar el mensaje *readyToFly*, lo que fuerza a los esclavos a despegar. La fase inicial termina cuando todos los drones alcanzan su posición en la formación de vuelo, cambiando al estado *Setup finished*.

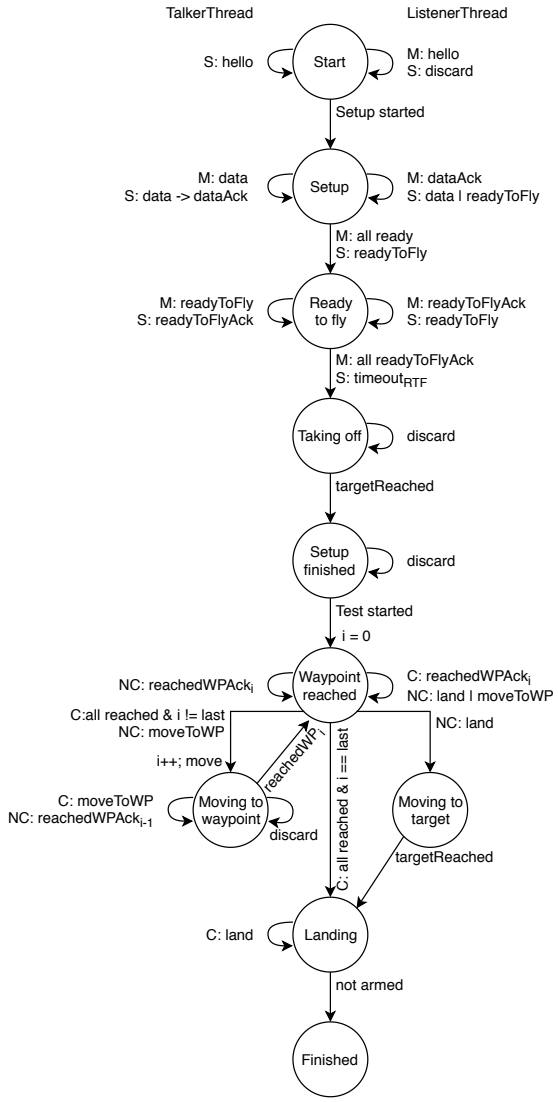


Fig. 1. MUSCOP: Máquina de estados finitos.

El verdadero protocolo de coordinación de vuelo comienza cuando el usuario pulsa un botón para comenzar el vuelo. La posición inicial de cada dron se considera el primer waypoint de la misión, por lo que todos comienzan en el estado *Waypoint reached*. Cuando el dron maestro recibe el mensaje *reachedWPAck* de todos los esclavos, empieza a moverse al siguiente waypoint (estado *Moving to waypoint*), y obliga a los esclavos a seguirle con el mensaje *moveToWP*. Todos los drones continúan en el estado *Moving to waypoint* hasta que llegan al siguiente waypoint. Durante el proceso, el dron maestro continua enviando la orden de moverse a dicho waypoint, mientras que los esclavos envían el reconocimiento de haber alcanzado con anterioridad el anterior waypoint. Este comportamiento redundante aumenta la fiabilidad del protocolo, ya que los mensajes entre drones se podrían perder por la distancia o la presencia de ruido en el canal de comunicación. Además, se puede afirmar que MUSCOP se caracteriza por una ocupación muy baja del canal, pues los mensajes transmitidos durante el vuelo son muy cortos (6 y 14 bytes, respectivamente). Cuando todos los drones llegan al último waypoint de la misión, el maestro aterriza en su ubicación actual y envía el mensaje *land*, que incluye su ubicación, y obliga a los esclavos a iniciar también el proceso de aterrizaje. Antes de aterrizar, los esclavos se aproximan al maestro manteniendo la misma formación de vuelo, pero reduciendo la distancia entre ellos a solamente 5 metros para reducir el área de aterrizaje del enjambre. Esta estrategia evita aterrizar en sitios no previstos o incluso peligrosos, como sucedería si los drones no se reagrupasen, lo que podría hacer difícil recuperarlos después.

C. Formato de los mensajes

La figura 2 muestra la estructura de los mensajes transmitidos del dron maestro a los esclavos y viceversa. En esta sección se detalla su contenido y propósito. Todos los mensajes comienzan con el campo *type*, que identifica el tipo de mensaje, y son enviados periódicamente (periodo de 200 ms) debido a que puede haber pérdida de mensajes en el canal de comunicación. Además, dado que las comunicaciones se basan en UDP broadcast, los mensajes enviados por el dron maestro son recibidos y procesados por todos los esclavos dentro de su radio de alcance, reduciendo al mínimo la posible sobrecarga de la red.

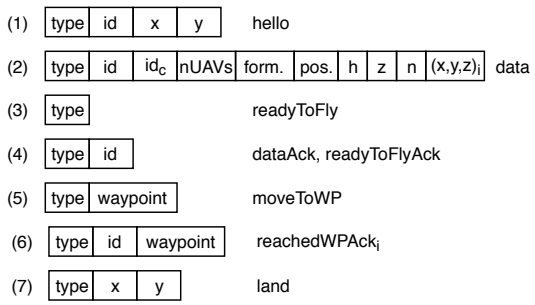


Fig. 2. MUSCOP: Tipos de mensajes.

El mensaje *hello* (1) lo envían los esclavos al maestro en cuanto son conectados. De este modo, el maestro detecta su presencia y determina su intención de formar parte del enjambre. El campo *id* representa un identificador único para cada dron, y se incluye en todos los mensajes que van dirigidos a un dron concreto. También se incluye la ubicación actual, lo que permite al maestro organizar la misión a seguir por cada esclavo.

Durante la fase *Setup*, el maestro determina la posición relativa de cada dron en la formación de vuelo y calcula su misión. El dron que está en el centro de la formación seguirá la misión original, mientras que el resto de drones seguirán una versión modificada que incluye un desfase constante respecto al maestro para mantener la misma posición en la formación durante todo el vuelo. El dron en el centro de la formación será el maestro, una decisión estratégica que optimiza las comunicaciones maestro-esclavo.

El mensaje *data* (2) incluye los siguientes campos:

- *id*. Identificador del dron al que va dirigido.
- id_c. Identificador del dron que será el maestro durante el vuelo, en el centro de la formación.

- *nUAVs*. Número de drones que formarán parte del la formación de vuelo. Parámetro requerido por el maestro para determinar si todos los drones han llegado a un waypoint.
- *form*. Tipo de formación entre las disponibles en ArduSim: lineal, matriz y circular.
- *pos*. Posición del dron al que va dirigido el mensaje, en la formación de vuelo.
- *h*. Orientación de la formación, fija durante todo el vuelo.
- *z*. Altitud sobre el suelo tras el despegue.
- *n*. Número de waypoints incluidos en el mensaje.
- $(x,y,z)_i$. Coordenadas de todos los waypoints incluidos en el mensaje.

El dron maestro envía el mensaje *readyToFly* (3) cuando comprueba que todos los esclavos han recibido el mensaje *data*, y hace que estén listos para iniciar el despegue.

Los mensajes *dataAck* y *readyToFlyAck* (4) son utilizados por los esclavos para informar al dron maestro que el correspondiente mensaje ha sido recibido. El primero indica que el dron ha recibido la misión a seguir, mientras que el segundo indica que está listo para despegar. Dado que el maestro necesita saber cuándo han recibido todos los mensajes, los dos indicados han de incluir el identificador del dron emisor.

El mensaje *moveToWP* (5) permite al dron maestro sincronizar la formación de vuelo. Una vez ha detectado que todos los esclavos han llegado al waypoint actual, éste envía este mensaje para encaminarlos al siguiente waypoint. Los esclavos utilizan el mensaje $reachedWPAck_i$ (6) para informar al maestro que ya han llegado al waypoint i.

Por último, el mensaje *land* (7) lo envía el maestro, incluyendo su ubicación, para que los esclavos determinen donde deben aterrizar, adoptando una versión más compacta de la formación de vuelo.

IV. Configuración de experimentos y métricas

En esta sección se incluyen detalles acerca de las misiones utilizadas para evaluar y validar MUSCOP. A continuación, se describe el proceso adoptado para determinar el error espacial y temporal asociado a cada experimento, los cuales representan las métricas utilizadas para validar MUSCOP.

A. Configuración de las misiones

El protocolo MUSCOP requiere una misión planificada como dato de entrada para que el dron maestro pueda guiar al enjambre a lo largo de la ruta definida. Hemos definido para los experimentos de validación varias misiones variando su complejidad. Para ello, se ha fijado la distancia de misión en 1840 metros, moviendo los drones hacia el noreste y aumentando el número de etapas (waypoints). En concreto, se han definido misiones con 2, 4, 6, 10, 14, 18 y 30 waypoints.

En la figura 3 se observan varios ejemplos de misión con la misma longitud, pero con distinto grado de complejidad: 2 waypoints (marrón), 6 waypoints (blanco) y 30 waypoints (verde), respectivamente.

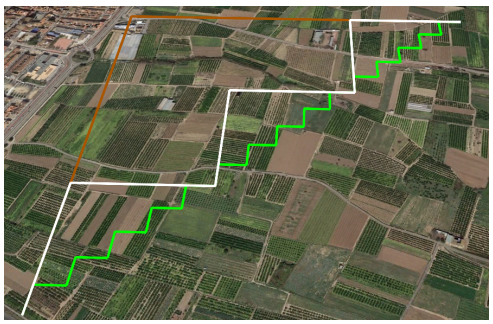


Fig. 3. Misiones ejemplo con 2 (marrón), 6 (blanco) y 30 waypoints (verde).

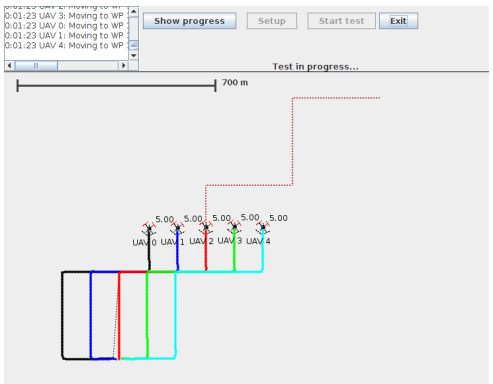


Fig. 4. Formación lineal de 5 drones en una misión de 6 waypoints en la plataforma de simulación ArduSim.

Se realizaron diversas simulaciones a partir de las misiones obtenidas. La figura 4 ilustra una misión de 6 waypoints en ArduSim, así como un experimento en proceso con el protocolo MUSCOP para 5 drones.

Al terminar el experimento, ArduSim almacena para cada dron varios ficheros a partir de los que se puede obtener información recogida durante el vuelo: coordenadas UTM, velocidad, aceleración, altitud, orientación, etc. En la evaluación, la primera ubicación recogida durante el vuelo supone el inicio del experimento. Los datos recogidos se han interpolado a intervalos regulares y a lo largo del rango de tiempo durante el cual hay datos para todos los drones.

B. Métricas

A continuación se describe la metodología utilizada para calcular el error relativo de cada dron del enjambre al utilizar el protocolo MUSCOP. En general, se mide el retraso temporal producido entre el dron maestro y los esclavos. Para ello, primero es necesario determinar el error general de la formación.

La figura 5 muestra distintos errores de posicionamiento para una formación lineal. Los puntos marcados con una "X" (color azul) representan la posición teórica en la formación calculada para cada esclavo a partir de la posición del maestro (dron rojo). Las líneas amarillas representan el error espacial, medido como la distancia entre la posición real del dron esclavo y la posición teórica ideal ("X" azul). Las fle-

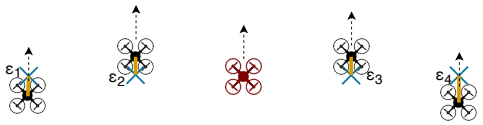


Fig. 5. Cálculo del error espacial en la posición relativa maestro-esclavo.

chas negras representan la dirección de movimiento de cada dron en el enjambre.

Se define el error general de la formación en el instante i (ε_i) como la suma del error espacial de todos los drones en la figura:

$$\varepsilon_i = \sum_{k=1}^{n} \varepsilon_{k_i} \tag{1}$$

El error o retraso espacial para un esclavo en concreto en el instante i (ε_{k_i}) se obtiene como la distancia entre la ubicación teórica en la formación $\overrightarrow{T_{k_i}}$ y la posición real $\overrightarrow{T'_{k_i}}$:

$$\varepsilon_{k_i} = \sqrt{(x_{k_i} - {x'}_{k_i})^2 + (y_{k_i} - {x'}_{k_i})^2} \tag{2}$$

Donde:

$$\overrightarrow{T_{k_i}} = (x_{k_i}, y_{k_i}) = \overrightarrow{T_{M_i}} + \overrightarrow{\Delta_k}$$
$$\overrightarrow{T'_{k_i}} = (x'_{k_i}, y'_{k_i})$$

Los valores ${x'}_{k_i}$ y ${y'}_{k_i}$ corresponden a la coordenada real obtenida durante el experimento para los ejes x e y, respectivamente, para cada esclavo en el instante i, a partir del conjunto de datos interpolado.

Teniendo en cuenta la ubicación actual del dron maestro $\overrightarrow{T_{M_i}}$ en el instante i, la posición teórica en la formación $\overrightarrow{T_{k_i}}$ se obtiene añadiendo el desfase correspondiente a cada esclavo $\overrightarrow{\Delta_k}$:

$$\overrightarrow{\Delta_k} = (\delta x_k \cos(h) + \delta y_k \sin(h), \delta y_k \cos(h) - \delta x_k \sin(h)) \tag{3}$$

El desfase teórico entre maestro y esclavo permanece constante para cualquier instante i, y depende de la orientación de la formación h y del desfase específico para cada dron respecto al dron maestro: $\delta_k = (\delta x_k, \delta y_k)$.

Por último, el retraso o error temporal respecto a la formación se calcula como $\varepsilon_{k_i}/v_{k_i}$, donde v_{k_i} es la velocidad actual del dron k.

V. Evaluación de prestaciones y validación

A continuación se procede a validar el protocolo propuesto por medio de un extenso conjunto de experimentos variando el número de drones del enjambre, la distancia entre los mismos, y la complejidad de la misión. La evaluación se divide en 5 partes: (i) impacto de la complejidad de la misión; (ii) retraso de misión debido a MUSCOP; (iii) impacto de la pérdida de mensajes; (iv) impacto al variar la distancia entre drones; y por último (v) análisis de escalabilidad. En todos los experimentos realizados se ha descartado la información del despegue y aterrizaje de las trazas, analizando el funcionamiento de MUSCOP solamente durante la fase de vuelo.

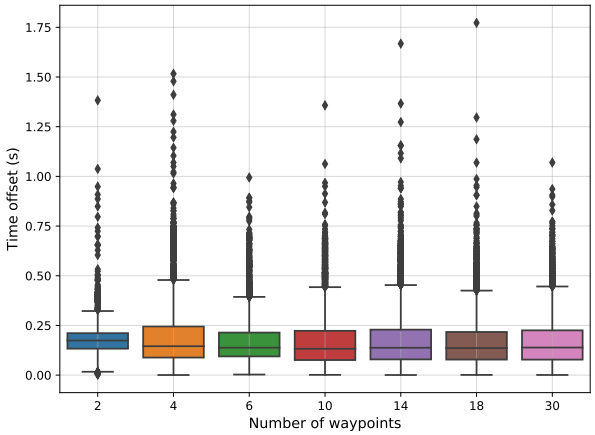


Fig. 6. Retraso temporal con 9 drones (canal ideal).

A. Impacto de la complejidad de la misión

En esta sección se evalúa el error espacial y temporal introducido en la formación por el protocolo propuesto. Los experimentos se realizaron con la configuración por defecto del simulador ArduSim, a una velocidad planificada de *10 m/s*, usando una formación lineal de 9 drones, utilizando un canal de comunicación ideal (sin pérdidas), y variando la complejidad de la misión. Esta complejidad se define variando la cantidad de waypoints, pero manteniendo la longitud total de la misión, tal y como se detalla en la sección IV-A. En total, el experimento con cada tipo de misión se repitió cinco veces, con una distancia teórica entre drones de 50 metros.

La figura 6 muestra los resultados obtenidos al variar la complejidad de la misión. Se observa que el error temporal medio es de 200 ms, un valor muy bajo. También se observan valores atípicos cuyo rango varía ligeramente en función de la complejidad de la misión.

En lo que respecta al error espacial, la figura 7 muestra que, en general, tiende a disminuir cuando la misión se vuelve más compleja. Para comprender la causa de estos resultados, hay que considerar que, a medida que la distancia entre waypoints disminuye al aumentar la complejidad, los drones tienen menos tiempo para acelerar y no llegan a alcanzar la velocidad planificada, produciendo por lo tanto un error en distancia inferior para un mismo retraso temporal.

Se concluye que, en un canal de comunicaciones ideal, la complejidad de la misión del enjambre, medida como el número de waypoints que la forman, es un factor que afecta directamente al error en distancia de la formación cuando los drones no tienen tiempo suficiente para alcanzar la velocidad máxima, aunque es irrelevante en cualquier otro caso. Además, el retraso temporal de cada dron es similar a la frecuencia de actualización del protocolo respecto al envío de mensajes (200 ms), lo que significa que el funcionamiento del protocolo es prácticamente ópti-

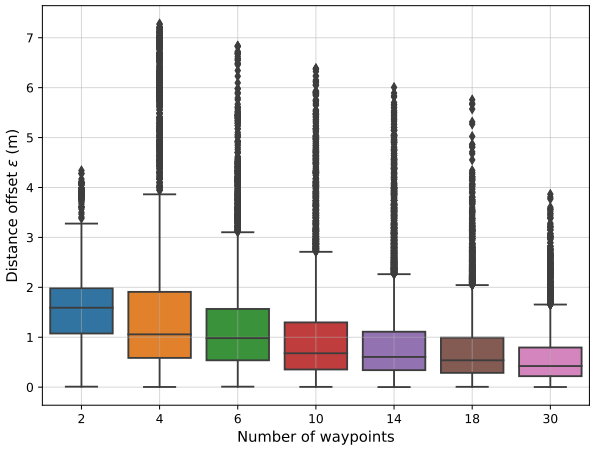


Fig. 7. Retraso espacial con 9 drones (canal ideal).

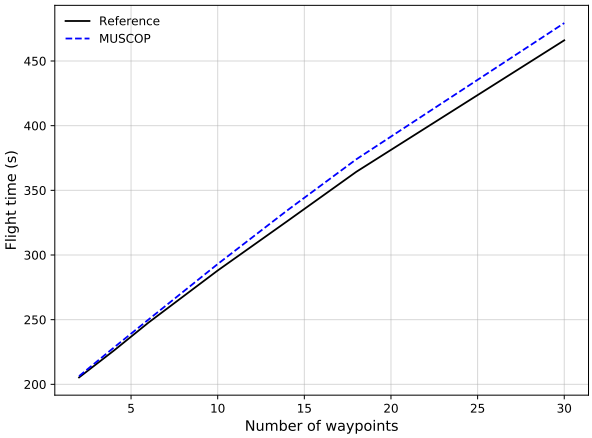


Fig. 8. Retraso temporal con 9 drones (canal ideal).

TABLA I
RETRASO DE TIEMPO DE VUELO.

Número de waypoints	Misión referencia (s)	Misión MUSCOP (s)	Δt (s)
2	205.33	206.33	1.00
4	226.00	228.33	2.33
6	247.67	250.00	2.33
10	288.00	293.00	5.00
14	326.00	334.33	8.33
18	364.33	374.00	9.67
30	466.00	479.33	13.33

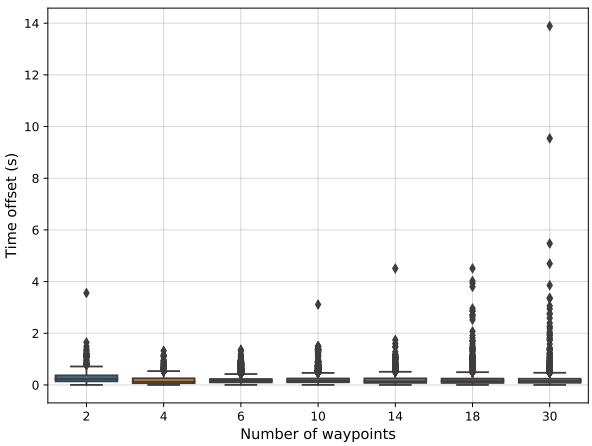


Fig. 9. Retraso temporal con 9 drones (canal con pérdidas).

mo.

B. Retraso de misión

En esta sección se analiza la sobrecarga temporal o retraso en el tiempo de misión que introduce MUSCOP. Para ello se midió el tiempo de vuelo variando el número de waypoints (ver tabla I), obteniendo el valor medio de 5 repeticiones por experimento. A continuación se comparó el tiempo obtenido con el que necesita un único dron para completar la misión de forma independiente (*Misión referencia*), obteniendo el retraso total durante el vuelo *(Δt)*. Se observa que el protocolo añade al tiempo de vuelo un retraso aproximado de 0.55 segundos por cada waypoint recorrido, lo que puede considerarse un retraso bastante reducido.

La figura 8 muestra que el tiempo de vuelo aumenta linealmente con el número de waypoints, y que dicho parámetro es el realmente dominante, mientras que el retraso introducido por MUSCOP es prácticamente despreciable en términos relativos.

C. Impacto de la pérdida de mensajes

En las secciones anteriores se evaluó el funcionamiento de MUSCOP en un canal de comunicaciones ideal para determinar con precisión el impacto de la complejidad de la misión y del protocolo en el tiempo de vuelo. A continuación se procede a evaluar MUSCOP con un canal de comunicación más realista, basado en la tecnología IEEE 802.11a, donde el rango de transmisión es de aproximadamente 1300 metros (condiciones del canal basadas en experimentos reales [14]). Para los experimentos se mantuvieron las mismas condiciones de las secciones anteriores, midiendo específicamente el impacto de la pérdida de mensajes en el funcionamiento del protocolo. Las figuras 9 y 10 muestran el retraso temporal y espacial obtenido. En la primera se observa que, en general, el retraso no varía significativamente respecto a los resultados obtenidos con el canal ideal (ver figura 6), aunque en estos resultados se observan valores atípicos significativamente mayores. Además, los valores atípicos son mayores al aumentar la complejidad de la misión, como sería de esperar. En la figura 10 se comprueba que el error medio en distancia disminuye al aumentar la complejidad de la misión. Como ya se ha indicado con anterioridad, esto se debe a que el dron no puede alcanzar la velocidad máxima planificada.

En la tabla II se puede comparar el comportamiento del protocolo usando un canal ideal y uno con pérdida de mensajes, para la formación analizada, y obteniendo el valor medio de todo el experimento para todos los drones. En general, resulta evidente que la diferencia entre el retraso temporal con canal ideal y con pérdida de mensajes es en realidad mínima, lo que valida el protocolo MUSCOP bajo condiciones realistas.

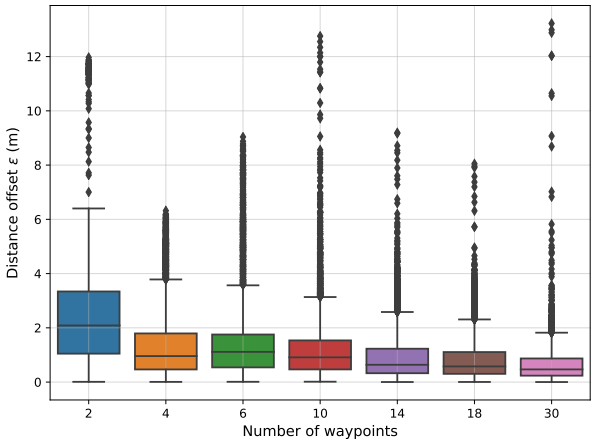


Fig. 10. Retraso espacial con 9 drones (canal con pérdidas).

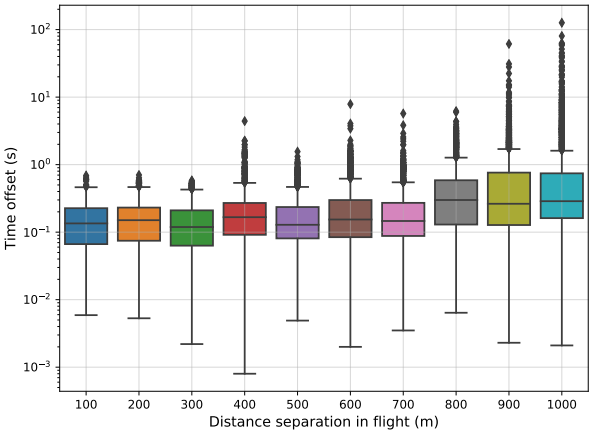


Fig. 11. Retraso temporal vs. distancia maestro-esclavo.

TABLA II
COMPARACIÓN CANAL IDEAL VS. CANAL CON PÉRDIDAS.

Número de waypoints	Retraso espacial (m)		Retraso temporal (s)	
	Canal ideal	Canal con pérdidas	Canal ideal	Canal con pérdidas
2	1.5457	2.3802	0.1757	0.2685
4	1.5700	1.4731	0.1993	0.1854
6	1.2422	1.4785	0.1728	0.2076
10	1.0139	1.2006	0.1715	0.1996
14	0.9051	0.9471	0.1777	0.1843
18	0.7483	0.8259	0.1627	0.1855
30	0.5806	0.6306	0.1643	0.1829

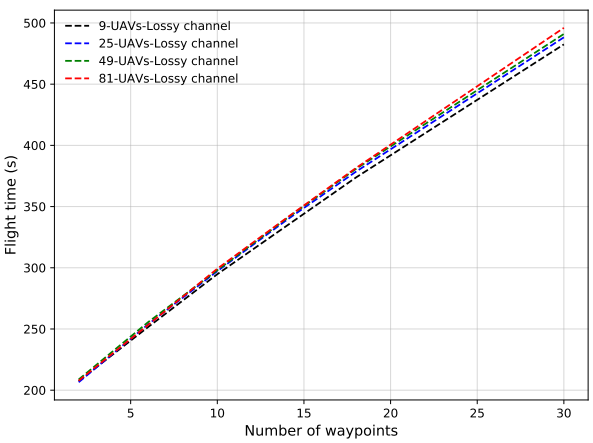


Fig. 12. Análisis de escalabilidad: tiempo de vuelo vs. número de drones y complejidad de misión.

D. Impacto al variar la distancia entre drones

Se puede observar en el análisis anterior que la distancia entre drones afecta al error de la formación, tanto en términos de retraso temporal como de posición relativa maestro-esclavo. En esta sección se profundiza en el estudio del efecto que produce este factor en la cohesión de la formación, y por lo tanto la pérdida de mensajes en el canal de comunicación. Para ello se varió la distancia de los esclavos al maestro desde 100 hasta 1000 metros. Además, la formación constó de 3 drones (1 líder y 2 esclavos), por lo que la distancia de los esclavos al maestro es siempre la misma. El número de waypoints de la misión fue 14, y el experimento se repitió 5 veces para cada distancia testada.

La figura 11 muestra el retraso temporal al variar la distancia entre drones. En general se observa que el valor medio aumenta a medida que la separación entre drones también aumenta. Éste es el comportamiento esperado en un canal de comunicación con pérdidas, donde una distancia elevada dificulta la sincronización entre drones, lo que se convierte en un problema crítico. También cabe mencionar que, hasta una distancia maestro-esclavo de 300 metros, el retraso temporal no supera el segundo, un valor bastante aceptable.

E. Análisis de escalabilidad

Por último se analiza el efecto que produce en MUSCOP el aumentar el número de drones y la complejidad de la misión. Para ello se realizaron experimentos con 9, 25, 49 y 81 drones. La distancia entre el maestro y el esclavo más alejado se mantuvo constante en todos los experimentos (106 m), con objeto de mantener el mayor grado posible de similitud entre los mismos.

La figura 12 muestra los resultados de escalabilidad al variar el número de drones usando un canal de comunicación con pérdidas. Se observa que, en general, el número de drones no tiene mucho impacto, lo que significa que la solución propuesta es muy escalable y es capaz de sincronizar eficientemente todos los drones que forman el enjambre. La figura también muestra que el número de waypoints es el factor con mayor impacto en el tiempo de vuelo. En concreto, se observa que la misión más compleja (30 waypoints) hace crecer el tiempo de vuelo a unos 450 segundos, más del doble si se compara con la misión que consta de solamente 2 waypoints.

VI. CONCLUSIONES Y TRABAJO FUTURO

En la actualidad, el despliegue y la gestión eficiente de enjambres de drones es un campo de investi-

gación que está ganando atención debido al extenso número de aplicaciones que están surgiendo, como sensorización remota, misiones de rescate, o despliegue aéreo de repetidores de datos multi-salto, entre muchos otros usos. Por lo tanto, es un reto interesante desarrollar estrategias de control para manejar adecuada y eficientemente enjambres de grandes dimensiones, a pesar de la pérdida de mensajes que se produce en el canal de comunicación inalámbrico.

En este trabajo se propone MUSCOP, una solución que gestiona eficientemente la sincronización de los drones de un enjambre mientras realiza una misión planificada conjunta. Para ello, primero se incluye una definición formal del protocolo por medio de una máquina de estados finitos, detallando los mensajes transmitidos, y después se valida la solución mediante la plataforma de simulación ArduSim. Los resultados experimentales muestran la efectividad de MUSCOP manteniendo la cohesión del enjambre bajo distintas condiciones, adaptándose bien a la pérdida de mensajes, y escalando fácilmente con el número de drones sin una penalización significativa en su funcionamiento. De hecho, los experimentos muestran que el factor que más afecta al tiempo de vuelo es la complejidad de la misión, independientemente del número de drones incluidos en el enjambre. Además, los experimentos también indican que, en un canal de comunicación ideal, el tiempo de vuelo solamente aumenta 0.55 segundos por waypoint debido a las necesidades de sincronización del protocolo, un valor que se incrementa muy ligeramente al utilizar un canal de comunicación con pérdida.

Como trabajo futuro se prevé validar el protocolo propuesto con distintas formaciones. Dado que ArduSim permite desplegar el protocolo implementado directamente en drones reales, también se prevé realizar una campaña intensiva de experimentos combinando quadcópteros y hexacópteros para comparar los resultados con los obtenidos en simulación. Además, también se desarrollará un nuevo protocolo para permitir despegar un elevado número de drones de forma eficiente y segura.

Agradecimientos

El presente trabajo ha sido financiado por el "Ministerio de Ciencia, Innovación y Universidades, Programa Estatal de Investigación, Desarrollo e Innovación Orientada a los Retos de la Sociedad, Proyectos I+D+I 2018", España, subvención: RTI2018-096384-B-I00.

Referencias

[1] B. S. Faiçal, G. Pessin, G. P. R. Filho, A. C. P. L. F. Carvalho, G. Furquim, and J. Ueyama, "Fine-tuning of UAV control rules for spraying pesticides on crop fields," in *2014 IEEE 26th International Conference on Tools with Artificial Intelligence*, Nov 2014, pp. 527–533.

[2] D. Albani, J. IJsselmuiden, R. Haken, and V. Trianni, "Monitoring and mapping with robot swarms for agricultural applications," in *2017 14th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*, Aug 2017, pp. 1–6.

[3] Karen Anderson and Kevin J Gaston, "Lightweight unmanned aerial vehicles will revolutionize spatial ecology," *Frontiers in Ecology and the Environment*, vol. 11, no. 3, pp. 138–146, 2013.

[4] Patrick Vincent and Izhak Rubin, "A framework and analysis for cooperative search using UAV swarms," in *Proceedings of the 2004 ACM Symposium on Applied Computing*, New York, NY, USA, 2004, SAC '04, pp. 79–86, ACM.

[5] M. Aljehani and M. Inoue, "Multi-UAV tracking and scanning systems in M2M communication for disaster response," in *2016 IEEE 5th Global Conference on Consumer Electronics*, Oct 2016, pp. 1–2.

[6] European Commission, "Autonomous swarm of heterogeneous RObots for BORDER surveillance," `https://cordis.europa.eu/project/rcn/209949/factsheet/en`, Accessed: 2019-01-30.

[7] Francisco Fabra, Carlos T. Calafate, Juan-Carlos Cano, and Pietro Manzoni, "ArduSim: Accurate and real-time multicopter simulation," *Simulation Modelling Practice and Theory*, vol. 87, no. 1, pp. 170–190, sep 2018.

[8] Eric W Justh and Perinkulam S Krishnaprasad, "A simple control law for UAV formation flying," Tech. Rep., Maryland University Institute for Systems Research, 2002.

[9] R. L. Lidowski, B. E. Mullins, and R. O. Baldwin, "A novel communications protocol using geographic routing for swarming UAVs performing a search mission," in *2009 IEEE International Conference on Pervasive Computing and Communications*, March 2009, pp. 1–7.

[10] Achudhan Sivakumar and Colin Keng-Yan Tan, "UAV swarm coordination using cooperative control for establishing a wireless communications backbone," in *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: Volume 3 - Volume 3*, Richland, SC, 2010, AAMAS '10, pp. 1157–1164, International Foundation for Autonomous Agents and Multiagent Systems.

[11] Zhongliang Zhao and Torsten Braun, "Topology control and mobility strategy for UAV ad-hoc networks: A survey," in *Joint ERCIM eMobility and MobiSense Workshop*. Citeseer, 2012, pp. 27–32.

[12] Ilker Bekmezci, Ozgur Koray Sahingoz, and Şamil Temel, "Flying ad-hoc networks (FANETs): A survey," *Ad Hoc Networks*, vol. 11, no. 3, pp. 1254 – 1270, 2013.

[13] Ismail Bayezit and Barış Fidan, "Distributed cohesive motion control of flight vehicle formations," *IEEE Transactions on Industrial Electronics*, vol. 60, no. 12, pp. 5763–5772, 2013.

[14] F. Fabra, C. T. Calafate, J. C. Cano, and P. Manzoni, "A methodology for measuring UAV-to-UAV communications performance," in *2017 14th IEEE Annual Consumer Communications Networking Conference (CCNC)*, Jan 2017, pp. 280–286.

Delegación de Autorización Perimetral para Dispositivos IoT

Elías Grande y Marta Beltrán[1]

Resumen— **La gestión de control de acceso ofrece un gran número de retos dentro del Internet de las Cosas (IoT) dadas las limitaciones desde el punto de vista de cómputo, memoria, almacenamiento, ancho de banda y/o energía disponible en la mayoría de los dispositivos de bajo coste y dispositivos embebidos en el mundo físico. En este escenario, la computación perimetral (computación Fog o en la niebla) puede ser considerada como una oportunidad interesante para solventar los problemas de autorización, ya que desplegando dispositivos perimetrales cerca de los dispositivos IoT éstos son capaces de llevar a cabo lógicas intermedias y enrutado entre ellos y los recursos en el cloud. Este trabajo propone la delegación de autorización que disponen las capas perimetrales para los dispositivos IoT de recursos limitados usando especificaciones y protocolos como OAuth 2.0 y CoAP (Constrained Application Protocol). La solución propuesta está basada en dos diferentes roles que permiten el registro dinámico de los dispositivos IoT en el sistema así como la gestión de autorización en el consumo de recursos por dichos dispositivos IoT.**

Palabras clave— **Autorización, CoAP, Control de Acceso, Internet of Things, OAuth**

I. Introducción

Asegurar los niveles de seguridad apropiados ha llegado a ser un tema esencial para el éxito y la evolución del Internet de las Cosas (IoT). En este escenario, la protección de los recursos, servicios y aplicaciones ante accesos no autorizados todavía representa una de las oportunidades más grandes a ser abordada. El modelo tradicional de confianza basado en identificación, autenticación y autorización ha dejado de ser factible debido a la escalabilidad inherente de los proyectos IoT y a las frecuentes limitaciones de recursos de sus dispositivos. Es por eso que el modelo de seguridad requiere un nuevo enfoque teniendo en cuenta las consideraciones específicas de los escenarios IoT como la escalabilidad y la eficiencia, y basado preferiblemente en el conocimiento de las tecnologías ampliamente utilizadas a día de hoy apoyándose en los principios de mínimo nivel de privilegio y mínima superficie de exposición.

Los esquemas federados como OAuth pueden ser una solución excelente para abordar la autorización en escenarios IoT ya que permiten resolver problemas como la gestión de autorización limitada en tiempo, revocación de accesos, protección contra ataques de colusión, integración de algoritmos criptográficos cuando se requieren, etc. Sin embargo, la limitación de recursos hardware de la mayoría de dispositivos IoT les permite disponer solamente de un conjunto de funcionalidades ligeras que pueden ser soportadas, entre las cuales, no se suele incluir el soportar estos esquemas federados como OAuth o similares.

El nuevo paradigma de computación perimetral (computación Fog o en la niebla) introduce nodos clave (como son los smart gateways, controladores, etc.) al filo de la red, cerca de los dispositivos de recursos limitados para permitirles comunicarse con los recursos, servicios y aplicaciones en el cloud. Estos nuevos nodos pueden ser usados para llevar a cabo funciones de seguridad, evitar que los ataques se propaguen a un dominio IoT completo, controlar el alcance de las amenazas de seguridad, etc. En resumen, los dispositivos perimetrales se pueden usar como intermediarios lógicos o enrutadores entre los dispositivos físicos y las capas de Internet/Web para mejorar los niveles de seguridad.

Las principales contribuciones de este trabajo son (1) La especificación de un nuevo mecanismo de control de acceso para IoT en el cual la autorización es delegada por la capa perimetral (2) Un enfoque novedoso para manejar esta autorización en escenarios a gran escala con presencia de dispositivos con recursos limitados para el que se adapta y extiende la conocida especificación OAuth 2.0 (3) La definición de dos flujos sobre CoAP resolviendo los desafíos más relevantes que surgen en el escenario considerado: el registro en el sistema y el control de acceso (4) Una implementación completa de la especificación propuesta utilizada en un escenario real (en el contexto de agricultura inteligente), validando sus funcionalidades y evaluando sus niveles de eficiencia y seguridad.

El resto de este artículo se organiza como sigue. La Sección II proporciona una visión general del trabajo y tecnologías relacionadas. Teniendo todo esto en cuenta, esta sección también analiza las principales motivaciones para este trabajo. La Sección III describe las asunciones consideradas y presenta los dos flujos propuestos necesarios para resolver la autorización dentro del Internet de las Cosas con el enfoque de autorización delegada. La Sección IV detalla la validación y evaluación de la solución propuesta. Finalmente, la Sección V resume nuestras principales conclusiones y las líneas más interesantes para futuras investigaciones.

II. Antecedentes y Motivación

A. Autorización de Dispositivos en IoT

La autorización es la capacidad de controlar el acceso a los recursos, servicios y aplicaciones especificando los distintos privilegios de los diferentes

[1]E. Grande y M. Beltrán forman parte del Dpto. de Arquitectura de Computadores, ETSII, Universidad Rey Juan Carlos, Madrid, 28933 España e-mail: elias.grande@urjc.es y marta.beltran@urjc.es

agentes (usuarios, máquinas, etc.) a través de políticas de acceso. Esta gestión es un contexto muy desafiante en IoT ya que las soluciones tradicionales de autorización no resultan muy adecuadas debido a la escalabilidad, heterogeneidad, dinamismo, complejidad y, en la mayoría de los casos, a los recursos limitados de los dispositivos. Debe considerarse que los agentes involucrados en los procesos de autorización son a menudo usuarios que interactúan con los recursos solicitados utilizando dispositivos restringidos o directamente son estos dispositivos restringidos por sí mismos.

En esta clase de escenario se han propuesto diferentes soluciones. El primer grupo de ellas se apoyan en enfoques distribuidos y cooperativos, intentando una propuesta simple, ligera y eficiente de mecanismos de autorización que no consumen todos los recursos disponibles de los sensores y dispositivos IoT. Estos mecanismos se basan muy a menudo en técnicas criptográficas [1], [2], [3] o incluso en Blockchain [4], [5], [6].

El segundo grupo de estas soluciones se apoyan en enfoques centralizados y/o federados, confiando en algún tipo de motor o servidor de autorización para enriquecer las decisiones de control de acceso (basado en atributos del contexto, basado en riesgo, etc.). Estas soluciones intentan asignar casi toda la carga relacionada con estos complejos procesos de autorización a este motor o servidor, ejecutándose en una máquina con recursos ilimitados en comparación con los sensores y dispositivos IoT. Estos trabajos generalmente se basan en las políticas de ABAC (control de acceso basado en atributos) y XACML, o en tokens de acceso y OAuth. Buenos ejemplos pueden encontrarse en [7], [8], [9], [10], [11], [12] o [13].

Los desafíos más recurrentes abordados por trabajos basados en el uso de OAuth son el almacenamiento seguro de credenciales utilizadas para la autenticación de dispositivos y la gestión de consentimiento del usuario minimizando sus interacciones no automatizadas. Ambos son especialmente difíciles de resolver dada la limitación de los recursos disponibles.

El almacenamiento seguro de credenciales en los dispositivos IoT ha sido abordado de múltiples formas. Quizás las más representativas se basan en el uso de PUFs (Physically Unclonable Functions), que son funciones basadas en las características intrínsecas de los circuitos hardware de un dispositivo concreto capaces de construir mecanismos criptográficos con vistas a identificar de manera unívoca el dispositivo IoT [14], [15]. Otros trabajos proponen diferentes opciones aprovechándose de las características de seguridad proporcionadas por los sistemas operativos específicos de IoT como CerberOS [16] o apoyándose en claves criptográficas y certificados digitales [17]. Todos estos trabajos anteriores tienen algo en común: no abordan el registro de dispositivos IoT en la solución de autorización de manera automatizada, y el intercambio de credenciales siempre se realiza fuera de banda o es un problema que no se afronta explícitamente para dispositivos IoT restringidos sin capacidad de almacenamiento seguro.

Por otro lado, diferentes iniciativas se han focalizado en extender OAuth 2.0 con vistas a gestionar el consentimiento del usuario de una forma dinámica como la "IETF - OAuth2 Device Flow initiative" y la "Kantara - User Managed Access initiative". Estas iniciativas, al igual que otras similares, intentan definir cómo se supone que un usuario otorga su consentimiento a un dispositivo IoT con la menor interacción posible pero sin eliminar completamente esta interacción, es decir, no es algo automatizado. Por tanto, esta forma de gestionar los consentimientos sigue afectando la implementación de grandes proyectos de IoT porque aún se requiere la interacción manual del usuario.

B. Motivación y Casos de Uso

El objetivo principal de esta investigación es el de superar las limitaciones identificadas de los trabajos previos utilizando OAuth dentro del ámbito IoT. Más concretamente, la definición de un mecanismo de delegación de autorización basado en OAuth 2.0 para dispositivos restringidos IoT, que se centra en dispositivos que no son capaces de almacenar sus propias credenciales de forma segura. Además, nos centraremos en los entornos con una gran cantidad de estos dispositivos restringidos IoT, proponiendo una automatización completa de sus procesos de registro y autorización para garantizar niveles adecuados de escalabilidad.

Además, se abordará una limitación común de trabajos anteriores: como el enfoque de este trabajo son los dispositivos restringidos, las versiones habituales de OAuth no son adecuadas porque funcionan sobre HTTP y TLS. La solución propuesta en esta investigación está diseñada para funcionar sobre CoAP y DTLS, que es lo idóneo cuando se trata de dispositivos restringidos. Si bien el protocolo CoAP es similar al protocolo HTTP, al tratar de usar OAuth 2.0 sobre CoAP, deberán abordarse algunos problemas: la capa de transporte es UDP en lugar de TCP, la falta de redirecciones (no soportadas en el protocolo CoAP [18]), etc.

El enfoque seleccionado para lograr todos estos objetivos se apoya en la computación Fog, de borde o perimetral (smart gateways, dispositivos edge, etc.), por lo tanto, hay una gran cantidad de casos de uso que podrían beneficiarse de la solución propuesta: todos los escenarios IoT que requieren un alto grado de automatización de gestión, ya que incluyen grandes cantidades de dispositivos restringidos que trabajan sobre CoAP, pero también disponen de algunos dispositivos no restringidos (o al menos, menos restringidos) capaces de asumir el rol de los dispositivos de borde para realizar la delegación de autorización. Se pueden encontrar buenos ejemplos en Smart Cities, Industry 4.0, Smart Agriculture, Healthcare, etc.

III. Delegación de Autorización Perimetral

A. Arquitectura y Asunciones

Como se ha mencionado anteriormente, este trabajo propone un enfoque centrado en computación Fog, de borde o perimetral para resolver la autorización en IoT. Estos enfoques se basan principalmente en plataformas distribuidas en el borde de la red que sirven como un puente entre el mundo físico (sensores y actuadores, a menudo dispositivos muy restringidos) e Internet (servicios digitales ofrecidos en la nube).

En este trabajo se asume un modelo de tres capas (ver la figura 1), siendo los diferentes tres roles descritos a continuación:

- Servicios cloud: Estos servicios (o recursos o aplicaciones) ofrecen capacidades de cómputo, de almacenamiento, gestión y visualización de datos, etc. Son las entidades que necesitan autorizar a los dispositivos para que éstos puedan realizar ciertas tareas específicas. En este trabajo, se supone que los servicios en la nube admiten comunicaciones seguras a través de HTTPS y permiten la gestión de autorización basándose en el estándar OAuth 2.0. También se supone que estos servicios se ejecutan en servidores sin restricciones en términos de potencia de cómputo, memoria, almacenamiento, ancho de banda y/o energía.
- Dispositivos de borde (o dispositivos Edge): Estos dispositivos pueden ser gateways, controladores independientes o embebidos, agrupaciones de dispositivos de borde, etc. Los aspectos significativos que los convierten en dispositivos de borde son:
  1. Están ubicados cerca de dispositivos restringidos IoT y más lejos de los servicios cloud.
  2. Esta proximidad permite comunicaciones eficientes y de baja latencia con los dispositivos restringidos.
  3. Pueden almacenar información confidencial de manera local, como claves, credenciales o tokens.
  4. Actúan como intermediarios y/o enrutadores de los dispositivos restringidos cuando éstos dispositivos necesitan interactuar con los servicios cloud.

  Se asume que los dispositivos de borde admiten comunicaciones a través de HTTPS (para comunicarse de manera segura con servicios cloud) y CoAP sobre DTLS [19] (para comunicarse de manera segura con los dispositivos restringidos). También deben poder custodiar de manera segura las credenciales de OAuth 2.0, y negociar con los servicios cloud para el acceso y la actualización de los tokens con diferentes scopes cuando sea necesario.
- Dispositivos restringidos IoT: Sistemas empotrados en el mundo físico con recursos muy limitados. Estos dispositivos no son compatibles con OAuth 2.0, y se asume que no tienen almacenamiento seguro para custodiar ningún tipo de secreto o credencial. También se asume que sólo pueden ejecutar funciones criptográficas muy ligeras, ya que si estas funciones son demasiado pesadas, pueden provocar la extenuación de los recursos existentes y/o un consumo de energía inaceptable. Finalmente, se debe tener en cuenta que, como ya se mencionó, se supone que los dispositivos restringidos pueden comunicarse con los dispositivos perimetrales o de borde utilizando CoAP sobre DTLS [19].

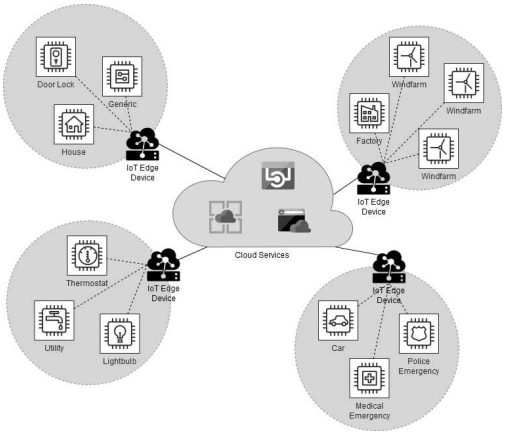


Fig. 1. Modelo de tres capas considerado en este trabajo.

Dada esta arquitectura y supuestos, la solución de delegación de autorización que se propone en las siguientes secciones especifica cómo los dispositivos restringidos pueden interactuar con los servicios cloud a través de los dispositivos perimetrales que admiten OAuth 2.0 y cómo éstos pueden delegar su autorización a los diferentes dispositivos restringidos IoT.

Inicialmente se requieren dos flujos diferentes. Primero, se debe definir un flujo de registro de dispositivos para permitir que un dispositivo perimetral o de borde identifique al dispositivo restringido IoT (o a un grupo de ellos) que, en algún momento, requiera la delegación de autorización para interactuar con un servicio cloud. Este flujo debería permitir que los dispositivos restringidos negocien sus propios scopes de acceso.

En segundo lugar, se requiere un flujo de acceso a los recursos protegidos para especificar cómo un dispositivo restringido IoT consume servicios cloud a través de un dispositivo de borde y cómo, si el dispositivo restringido crea un recurso dentro del cloud, éste vinculará un identificador único de este recurso a la identificación del dispositivo restringido. Gracias a este enlace entre la identificación del dispositivo restringido y el identificador del recurso creado, sólo ese dispositivo restringido concreto, podrá acceder, actualizar o eliminar sus propios recursos.

B. Flujo de Registro

El objetivo de este flujo es doble: hacer frente a las limitaciones de los recursos del dispositivo IoT y permitir una alta escalabilidad. Por un lado, este flujo

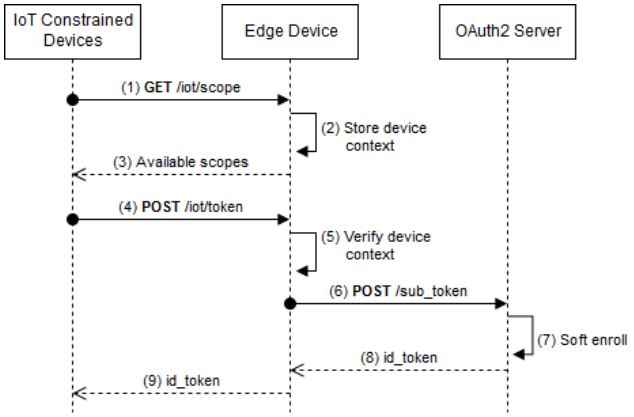


Fig. 2. Flujo de Registro.

evita la necesidad de realizar una autenticación robusta que consumiría todos los recursos disponibles (y limitados) del dispositivo restringido IoT. Con el fin de identificar y autenticar este dispositivo, este trabajo propone el uso de un token de identidad emitido después de la validación de una huella digital construida a través de atributos relacionados con el contexto del dispositivo (estáticos y/o dinámicos). El token de identidad funciona como un identificador único y opaco que representa el contexto del dispositivo almacenado en el cloud durante su registro y siempre se valida cuando el dispositivo restringido intenta crear, acceder, actualizar o eliminar un recurso en el cloud. Por otro lado, evitar un modelo de autenticación robusto permite que un dispositivo de borde, incluso con recursos limitados también, registre y administre una gran cantidad de dispositivos restringidos IoT.

La figura 2 muestra el flujo de registro propuesto que consta de los siguientes pasos:

1. El dispositivo restringido IoT (IoT constrained device) solicita al dispositivo de borde (Edge device) qué scopes de OAuth 2.0 tiene disponibles. Esta petición debe ser CoAP, posiblemente un GET o un POST dependiendo de la implementación del dispositivo de borde correspondiente.
2. El dispositivo de borde recupera de esta solicitud y registra todos los atributos que describen el contexto actual (la huella digital mencionada anteriormente) del dispositivo restringido IoT. Si la solicitud era un CoAP GET, el contexto incluirá la dirección lógica del dispositivo restringido IoT y su user-agent, dependiendo de las cabeceras incluidas en la implementación de CoAP. Si la solicitud era un POST CoAP, el cuerpo del mensaje también podría incluir la dirección física del dispositivo, su ubicación geográfica, etc.
3. El dispositivo de borde responde al dispositivo restringido IoT con el conjunto de scopes de OAuth 2.0 de los que dispone.
4. Una vez el dispositivo restringido IoT elige entre los scopes disponibles cual cubre mejor sus necesidades, éste solicita un token de identidad válido sólo para el scope específico al dispositivo de borde.
5. El dispositivo de borde valida la solicitud comprobando que el scope solicitado está soportado y que el contexto recuperado de la nueva solicitud coincide con el de la solicitud registrada previamente.
6. Si el scope es válido y el contexto se verifica con éxito, el dispositivo de borde solicita al servidor OAuth 2.0 la delegación de autorización basada en su propio token de acceso. En este momento, el servidor de autorización OAuth 2.0 vincula la identidad del dispositivo restringido IoT específico con el token de acceso del dispositivo de borde para generar un nuevo token de identidad que representa la nueva autorización otorgada al dispositivo restringido IoT. Esta delegación puede realizarse tantas veces como sea necesario con todos los dispositivos que inician el flujo de registro a través del mismo dispositivo de borde. Esta jerarquía de delegación permite a los servicios cloud la capacidad de revocar todos los tokens de identidad vinculados a un token de acceso concreto, revocando sólo este token de acceso de la misma manera que una infraestructura de clave pública (PKI) funciona con las autoridades de certificación intermedias y los certificados emitidos por ellas. En este paso, se supone que el token de acceso se ha negociado previamente entre el dispositivo de borde y el servidor OAuth 2.0 de forma segura siguiendo la especificación tradicional.
7. El servidor OAuth 2.0 realiza un registro del dispositivo restringido en función de su contexto (propagado por el dispositivo de borde) y vincula este contexto al token de acceso del dispositivo de borde. Después de eso, se genera el token de identidad para el dispositivo restringido IoT con un tiempo de caducidad corto (que varía de un dominio de aplicación a otro pero que no debe exceder una hora) estando dicho token también vinculado al token de acceso del dispositivo de borde.
8. El token de identidad generado se envía de nuevo al dispositivo de borde. Como en cualquier otro modelo de control de acceso basado en tokens que proporcionan permisos para acceder a recursos protegidos, el token debe transmitirse siempre a través de un canal seguro.
9. Finalmente, el token de identidad se envía al dispositivo restringido IoT (de nuevo a través de un canal seguro) y, por lo tanto, se concluye el procedimiento de registro en este dispositivo de borde.

C. Flujo de Acceso

Este flujo especifica cómo un dispositivo restringido IoT, identificado a través de su token de identidad obtenido durante el flujo de registro en un dispositivo de borde específico, está autorizado para realizar un acceso a un recurso, servicio o aplicación

cloud. El token de identidad es una herramienta poderosa para resolver dos aspectos fundamentales en cualquier flujo de autorización basado en una delegación: el tiempo de vida (TTL) del token vinculado al acceso solicitado y la trazabilidad o no repudio de los accesos realizados.

Debe destacarse que el propio token de acceso de OAuth 2.0 del dispositivo de borde dispone de por sí de su propio tiempo de expiración. Más allá de este tiempo, dicho token de acceso no será aceptado por los servicios cloud por lo que si el dispositivo de borde aún necesita acceder a ellos, éste debe renovar su token de acceso. Además, este tiempo de caducidad está relacionado con los accesos del propio dispositivo de borde y no con los tiempos de expiración de los tokens de identidad generados a través de la autorización delegada, que podrían ser más breves.

En este trabajo, el TTL se define como la cantidad de tiempo que una delegación específica de autorización realizada a través de un dispositivo de borde permite que un dispositivo IoT cree, acceda, actualice o elimine en los servicios cloud cualquier recurso creado con el token de identidad obtenido durante el flujo de registro correspondiente a esa delegación. Cuando este token de identidad caduque o cuando se revoque (por ejemplo, debido a que el contexto del dispositivo restringido cambia en un grado significativo), el dispositivo restringido IoT no podrá acceder a esos recursos más, incluso si genera un nuevo token de identidad ya que este token se considerará como una delegación de autorización diferente. Este funcionamiento puede entenderse como una zona aislada con fecha de caducidad creada por el dispositivo de borde en el servicio en la nube para permitir que un dispositivo restringido IoT específico funcione durante el TTL con su token de identidad asociado (y, por lo tanto, con el contexto específico del dispositivo restringido durante su registro).

Esta función de aislamiento con fecha de caducidad proporciona capacidades de trazabilidad y no repudio ya que un token de identidad se asigna de manera unívoca al contexto de un dispositivo restringido IoT y, por lo tanto, a una delegación específica de autorización a través de un dispositivo de borde y su token de acceso. Esta función permite saber qué dispositivo está haciendo qué sobre cada servicio cloud en cualquier momento.

El flujo de acceso se ilustra en la figura 3, siguiendo los siguientes pasos:

1. El dispositivo restringido IoT (IoT constrained device) realiza una solicitud al dispositivo de borde (Edge device) a través del que se ha registrado vía POST CoAP incluyendo dos cabeceras CoAP: Proxy-Uri y ETag. La cabecera Proxy-Uri se refiere al uri del servicio cloud donde el dispositivo restringido necesita realizar el acceso con el dispositivo de borde como intermediario. Por otro lado, la cabecera ETag incluye el token de identidad del dispositivo restringido IoT obtenido durante el flujo de registro. Esta cabecera se usa para enviar dicho token de identidad porque, por definición, un ETag es un identificador opaco asignado a una versión específica de un recurso. Dado que el acceso del dispositivo restringido IoT a sus propios recursos protegidos cloud sólo debe ser permitido por este token de identidad mientras no esté caducado o revocado y ningún otro token de identidad podría permitir el acceso a esos recursos, esta cabecera que se describe en la RFC del protocolo CoAP [18] encaja perfectamente con el propósito de la definición de token de identidad explicada en este trabajo y, por lo tanto, no es necesario definir una nueva cabecera CoAP.
2. Este paso no es obligatorio y depende del comportamiento del dispositivo de borde con respecto al registro del contexto del dispositivo restringido IoT. Si este contexto está disponible (almacenado en un caché, por ejemplo), este paso no es necesario. Si no lo está, el dispositivo de borde solicita la información de contexto al servidor OAuth 2.0 utilizando el token de identidad incluido en la cabecera ETag de la petición CoAP. Esta solicitud utiliza una nueva cabecera HTTP creada por este trabajo para cubrir esta necesidad específica: IoT-Forwarded.
3. Si se realizó el paso anterior y el token de identidad está asociado al token de acceso del dispositivo de borde que realiza la solicitud HTTP al servidor OAuth 2.0, éste recupera la información de contexto asociada al dispositivo restringido IoT que tiene almacenada y se la envía de vuelta al dispositivo de borde.
4. El dispositivo de borde verifica que el contexto del dispositivo restringido IoT que solicita acceder al servicio cloud coincida con la información de contexto disponible (almacenada en su propia caché o recuperada del servidor OAuth 2.0).
5. Si esta verificación es correcta, el dispositivo de borde traduce la petición CoAP a HTTP y realiza la solicitud a la uri incluida en la cabecera Proxy-Uri de la solicitud CoAP inicial. En esta solicitud al servicio cloud, también se requieren el token de identidad del dispositivo restringido

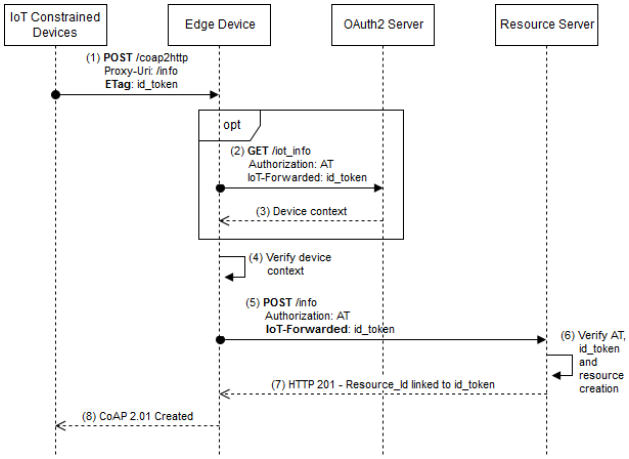


Fig. 3. Flujo de Acceso.

IoT y el token de acceso del dispositivo de borde. En el ejemplo mostrado en la figura 3, el POST CoAP es traducido a POST HTTP.

6. El servicio cloud primero valida el scope y el tiempo de caducidad del token de acceso presentado por el dispositivo de borde. Después de esta validación, debe comprobarse si el token de identidad está correctamente vinculado al token de acceso. Si todas estas verificaciones tienen un resultado positivo, y el método HTTP es un POST, el servicio cloud crea el recurso vinculado al token de identidad recibido. Cuando el método HTTP es un GET, un PUT o un DELETE, este paso es ligeramente diferente: el servicio cloud verifica que el token de identidad recibido es el mismo que creó el recurso la primera vez y, si esta verificación es correcta, permite al dispositivo restringido IoT acceder, actualizar o eliminar el recurso.
7. El servicio cloud responde al dispositivo de borde con una respuesta HTTP concordante a la petición que realizó en función de los resultados del paso anterior del flujo. Por ejemplo, en la figura 3 el servicio cloud produce una respuesta HTTP con el código de estado "HTTP 201 Created" y el identificador del recurso creado.
8. Finalmente, el dispositivo de borde traduce esta respuesta HTTP a la correspondiente respuesta CoAP para actuar como intermediario y propagar la información al dispositivo restringido IoT.

IV. Validación y Evaluación

A. Descripción del Caso de Uso

El IoT ofrece a los agricultores una herramienta poderosa para obtener un mejor control sobre los procesos, aumentando su eficiencia y calidad al tiempo que reduce los riesgos. En este caso de uso, nos centramos en una aplicación de monitoreo de ganado que consiste en el monitoreo remoto de vacas con mecanismos de alerta. El objetivo del proyecto es controlar la temperatura, la actividad y el comportamiento (movimiento de la cabeza, desplazamientos, nutrición, proximidad a otras vacas, etc.) de cada vaca individual utilizando collares de cuello inteligentes y no invasivos. Toda la información recopilada se carga en los servidores centrales para su almacenamiento, procesamiento y análisis; permitiendo a los agricultores iniciar varias estrategias para mejorar el cuidado y la productividad del ganado. Por ejemplo, los ciclos reproductivos de las vacas pueden modelarse para predecir las temporadas de apareamiento o el parto (para que el veterinario pueda prepararse, debe considerarse que las granjas ganaderas están muy lejos, en muchos casos, de los centros de población). Los ciclos de producción de leche también se pueden modelar, ajustando la dieta de las vacas para que sean beneficiosas para su salud y optimicen su producción.

Los sensores incluidos en los collares inteligentes son duraderos y fáciles de mantener, pero ofrecen recursos muy limitados. Esta es la razón por la que propusimos en este proyecto el uso de dispositivos de borde en cada establo de vacas y zonas de pastoreo al aire libre.

B. Implementación

La implementación de ambos roles, el del dispositivo de borde y el del dispositivo restringido IoT, se ha basado en el proyecto CoAPthon [20]. En dicho proyecto ha sido necesario desarrollar y añadir la funcionalidad de proxy CoAP a HTTP/HTTPS para cumplir con los requisitos del rol del dispositivo de borde descrito en las secciones anteriores de este artículo.

La implementación del dispositivo de borde, además de admitir la traducción de CoAP a HTTP y viceversa, permite una gestión adecuada del valor ETag (a partir de las cabeceras CoAP). Además, se ha incluido una caché simple en el dispositivo de borde para almacenar los diferentes contextos de dispositivos restringidos IoT. La implementación del dispositivo restringido IoT incluye la funcionalidad mínima requerida para probar los diferentes flujos propuestos en este trabajo.

Por otro lado, los servicios cloud que se ejecutan en los servidores, incluyendo el servidor de recursos (específico de la aplicación) y los complementos del servidor OAuth 2.0, se han desarrollado basándose en la especificación OAuth 2.0.

Con respecto a los complementos incluidos en el servidor OAuth 2.0, se deben hacer algunas consideraciones:

- Los servicios desarrollados para gestionar la delegación de autorización y todas las verificaciones relacionadas, así como otros servicios descritos en esta investigación, se han implementado en base a la especificación OAuth 2.0 sin afectar o modificar su implementación estándar.
- De la misma manera, el modelo de datos creado durante esta investigación amplía el modelo de datos de OAuth 2.0 (como se muestra en la figura 4) sin afectarlo ni modificarlo. Nuestra implementación solo agrega funcionalidades requeridas. La extensión mencionada incluye tres nuevas entidades: dispositivos, tokens de identidad y recursos. Los dispositivos representan el contexto de los dispositivos restringidos en el momento del registro en el sistema. Este contexto y la información del token de acceso del dispositivo de borde están vinculados al token de identidad generado. Cuando el dispositivo restringido crea un recurso, el identificador de recurso está vinculado a este token de identidad específico. Por lo tanto, este modelo de datos permite la trazabilidad de todos los recursos creados por dispositivos restringidos IoT gracias a la relación entre todas las entidades citadas sin ningún impacto en el modelo de datos OAuth 2.0 o en su comportamiento estándar.

Finalmente, se ha desarrollado un decorador para verificar tanto el token de identidad como su token

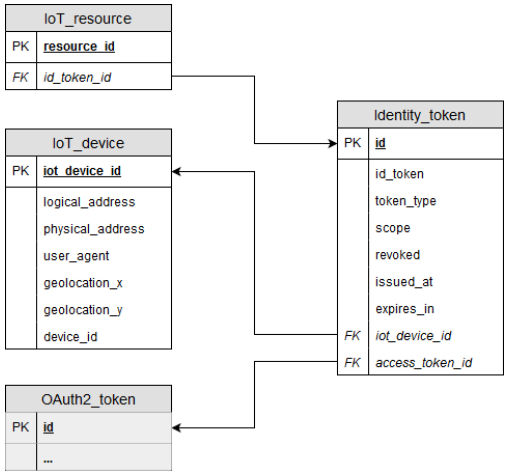


Fig. 4. Propuesta de extensión del modelo de datos de OAuth 2.0.

de acceso vinculado en los servicios cloud. Cuando el servidor de recursos quiere gestionar la delegación de autorización propuesta, las API del servidor de recursos deben protegerse anotándose con dicho decorador. Las decisiones tomadas por este decorador se pueden resumir en:

- Si la solicitud solo tiene un token de acceso, el decorador asume que no hay dispositivos restringidos involucrados en esta solicitud y no hace nada.
- Si la solicitud incluye un token de identidad y un token de acceso, y el método HTTP es un POST, el decorador verifica que el token de identidad no haya caducado o haya sido revocado. Después de esto, el decorador verifica si éste token de identidad está vinculado al token de acceso. Solo si estas verificaciones son exitosas, se genera el identificador del recurso vinculado al token de identidad y se carga en el contexto de la solicitud del servicio cloud para su uso dentro de esta ejecución y su posterior retorno al dispositivo de borde como identificador del recurso creado.
- Si la solicitud incluye un token de identidad y un token de acceso, pero el método HTTP es GET, PUT o DELETE, el decorador realiza las mismas verificaciones que en el caso anterior pero además, debe verificar si el identificador de recurso que solicitó el dispositivo restringido IoT está vinculado con el token de identidad incluido en la solicitud.

C. Evaluación de Escenarios y Discusión

La primera evaluación realizada en el caso de uso descrito se centra en la automatización del registro de los dispositivos restringidos IoT dentro del sistema altamente escalable propuesto sin ninguna interacción manual por parte del propietario de los recursos, ni en el registro ni en la gestión del consentimiento.

Por un lado, el flujo de registro propuesto cubre los sistemas altamente escalables sin interacción humana debido a que los dispositivos restringidos IoT hacen un auto registro basado en su propia huella digital del dispositivo y delegan en dispositivos o sistemas más potentes su propia autenticación y gestión de control de acceso en lugar de tratar de resolver por sí mismos dichos problemas. Esta delegación de la responsabilidad permite que los dispositivos restringidos dejen de lado la necesidad de un almacenamiento seguro u otras consideraciones de seguridad que obliguen a que sean más grandes o tengan un consumo excesivo de batería, y, por lo tanto, este modelo propuesto es relevante para la aplicación en sistemas IoT donde el tamaño del dispositivo o las restricciones del mismo, como el consumo de la batería o el peso del dispositivo, se convierten en problemas críticos.

Además, como el registro de los dispositivos restringidos IoT no requiere de interacción humana para la gestión del consentimiento en el flujo de registro, estos dispositivos pueden registrarse donde y cuando lo necesiten, incluso de forma itinerante, a través de establos y zonas de pastoreo al aire libre como ocurre en el caso de uso propuesto.

La segunda evaluación se realiza en base a un punto de vista de seguridad. Este punto de vista no se centró en establecer múltiples suposiciones para definir un modelo de arquitectura que evite los vectores de ataque a los que se expone un dispositivo restringido IoT de forma habitual; ataques físicos, ataques basados en protocolos de comunicación, ataques sobre los datos en reposo y ataques basados en el software del propio dispositivo IoT [21]. El supuesto principal consistía en asumir que todos estos vectores de ataque están ahí, y por lo tanto, el modelo de arquitectura debe abordarlos de manera segura, enfocándose en los principales problemas de privacidad generados por esos vectores de ataque en torno a los datos en sí: la minería de datos y la detección de intrusiones [22].

Por un lado, la gobernabilidad de los datos está gestionada por el propio modelo de autorización que evita por dise´ no la extracción de datos desde cualquier dispositivo restringido IoT comprometido:

- Primero, un dispositivo restringido IoT sólo puede acceder, en el mejor de los casos, a un subconjunto de todos los datos disponibles en los servicios cloud gracias a la propia delegación de autorización propuesta en este trabajo. Esto es así, porque el acceso de un dispositivo restringido siempre está vinculado y limitado por el scope del acceso autorizado de su dispositivo de borde asociado.
- Además, el subconjunto de datos disponible para cada dispositivo restringido IoT está limitado a su vez por el acceso basado en el tiempo a los recursos protegidos, la vida útil de los datos en sí y la zona de datos aislada definida para cada dispositivo restringido. Por un lado, el acceso basado en el tiempo y la vida útil de los datos limitan el acceso a los datos de un dispositivo restringido concreto, por lo que si un atacante compromete un dispositivo restringido,

sólo puede obtener datos reales del dispositivo comprometido por un corto espacio de tiempo. Por otro lado, la zona de datos aislada donde un dispositivo concreto almacena y recupera sus datos, impide que un atacante pueda recuperar la información de otro dispositivo restringido gracias a ataques de referencia directa de objetos u otros movimientos laterales ya que la zona de datos aislada no se comparte en ningún momento entre dispositivos restringidos.

Finalmente, gracias a las características comentadas anteriormente del modelo de autorización propuesto en este trabajo, el gobierno de datos definido permite vincular todos los datos generados por cada dispositivo restringido con su token de identidad utilizado en cada intervalo de tiempo. Esta asociación entre el identificador de recursos y el token de identidad permite mejorar la trazabilidad dentro del sistema para saber qué dispositivo y cuándo está accediendo a los datos. Por lo tanto, el sistema tiene la capacidad de detección de intrusos para aplicar contramedidas contra el propio dispositivo restringido o contra el dispositivo de borde asociado si fuera necesario.

V. Conclusiones

Este artículo ha propuesto una delegación de autorización para IoT basándose en un nuevo mecanismo de control de acceso federado usando tokens OAuth 2.0 y sobre el protocolo CoAP con vistas a permitir la autorización de dispositivos restringidos IoT a través de dispositivos de borde para el acceso a recursos cloud. El análisis realizado es un caso de uso de agricultura inteligente real donde se muestra cómo la solución propuesta es escalable (lo que permite el registro automático), es interoperable (se basa en los mismos conceptos del estándar OAuth 2.0, solo se extiende) y es suficiente para resolver el control de acceso en casi todos los escenarios con la eficiencia adecuada (en términos de consumo de recursos y latencias) y seguridad.

Ahora estamos trabajando para extender el mecanismo propuesto sobre MQTT (Message Queue Telemetry Transport) y para permitir el modo "roaming" cuando un dispositivo restringido IoT se reconecta a través de otro dispositivo de borde o perimetral diferente al suyo inicial.

Agradecimientos

Esta investigación ha sido parcialmente respaldada por el Gobierno de España (RTC-2017-6253-1) y por el Ericsson-URJC Chair ("Data Science applied to 5G").

Referencias

[1] S. Cirani and M. Picone, "Effective authorization for the web of things," in *2nd IEEE World Forum on Internet of Things*, 2015, pp. 316–320.

[2] A. Kurniawan and M. Kyas, "A trust model-based bayesian decision theory in large scale internet of things," in *IEEE 10th International Conference on Intelligent Sensors, Sensor Networks and Information Processing*, 2015, pp. 1–5.

[3] S. Sciancalepore, G. Piro, D. Caldarola, G. Boggia, and G. Bianchi, "On the design of a decentralized and multi-authority access control scheme in federated and cloud-assisted cyber-physical systems," *IEEE Internet of Things Journal*, 2018.

[4] A. Outchakoucht, H. ES-SAMAALI, and J. Philippe, "Dynamic access control policy based on blockchain and machine learning for the internet of things," *International Journal of Advanced Computer Science and Applications*, vol. 8, 01 2017.

[5] A. Ouaddah, A. Abou Elkalam, and A. Ait Ouahman, *Towards a Novel Privacy-Preserving Access Control Model Based on Blockchain Technology in IoT*, 09 2017, pp. 523–533.

[6] O. J. Pinno, A. Grégio, and L. C. Bona, "Controlchain: Blockchain as a central enabler for access control authorizations in the iot," in *2017 IEEE Global Communications Conference*, 2017, pp. 1–6.

[7] J. L. H. Ramos, M. P. Pawlowski, A. J. Jara, A. F. Gómez-Skarmeta, and L. Ladid, "Toward a lightweight authentication and authorization framework for smart objects," *IEEE Journal on Selected Areas in Communications*, vol. 33, pp. 690–702, 2015.

[8] P. Solapurkar, "Building secure healthcare services using oauth 2.0 and json web token in iot cloud scenario," in *Proceedings of the 2nd International Conference on Contemporary Computing and Informatics*, 2016, pp. 99–104.

[9] Q. Huang, L. Wang, and Y. Yang, "Decent: Secure and fine-grained data access control with policy updating for constrained iot devices," *World Wide Web*, vol. 21, 05 2017.

[10] S.-H. Lee, K.-W. Huang, and C.-S. Yang, "Tbas: Token-based authorization service architecture in internet of things scenarios," *International Journal of Distributed Sensor Networks*, vol. 13, 2017.

[11] F. Chen, Y. Luo, J. Zhang, J. Zhu, Z. Zhang, C. Zhao, and T. Wang, "An infrastructure framework for privacy protection of community medical internet of things," *World Wide Web*, vol. 21, no. 1, pp. 33–57, 2018.

[12] M. Beltrán, "Identifying, authenticating and authorizing smart objects and end users to cloud services in internet of things," *Computers & Security*, vol. 77, pp. 595–611, 2018.

[13] L. Cruz-Piris, D. Rivera, I. Marsá-Maestre, E. de la Hoz, and J. R. Velasco, "Access control mechanism for iot environments based on modelling communication procedures as resources," *Sensors*, vol. 18, no. 3, p. 917, 2018.

[14] F. Tehranipoor, N. Karimian, W. Yan, and J. Chandy, "Dram-based intrinsic physically unclonable functions for system-level security and authentication," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, pp. 1085–1097, 2017.

[15] M. N. Aman, K. C. Chua, and B. Sikdar, "Mutual authentication in iot systems using physical unclonable functions," *IEEE Internet of Things Journal*, vol. 4, no. 5, pp. 1327–1340, 2017.

[16] S. Akkermans, W. Daniels, G. Sankar R., B. Crispo, and D. Hughes, "Cerberos: A resource-secure os for sharing iot devices," in *Proceedings of the 2017 International Conference on Embedded Wireless Systems and Networks*, 2017, pp. 96–107.

[17] H. Kim and E. A. Lee, "Authentication and authorization for the internet of things," *IT Professional*, vol. 19, no. 5, pp. 27–33, 2017.

[18] K. H. Z. Shelby and C. Bormann, "The Constrained Application Protocol, rfc 7252," 2014, https://www.rfc-editor.org/info/rfc7252.

[19] H. Tschofenig and T. Fossati, "Transport layer security (tls) / datagram transport layer security (dtls): Profiles for the internet of things, rfc 7925," 2016, https://www.rfc-editor.org/info/rfc7925.

[20] G. Tanganelli, C. Vallati, and E. Mingozzi, "Coapthon: Easy development of coap-based iot applications with python," in *IEEE World Forum on Internet of Things*, 2015.

[21] H. A. Abdul-Ghani, D. Konstantas, and M. Mahyoub, "A comprehensive iot attacks survey based on a building-blocked reference model," *International Journal of Advanced Computer Science and Applications*, vol. 9, no. 3, 2018, http://dx.doi.org/10.14569/IJACSA.2018.090349.

[22] A. V. V. Zheng Yan, Peng Zhang, "A survey on trust management for internet of things," *Journal of Network and Computer Applications*, vol. 42, pp. 120–134, 2014.

Desarrollo de un sistema para medición y registro de RSSI en invernaderos

Dora Cama-Pinto[1], Miguel Damas[1], Juan Antonio Holgado-Terriza[2], Francisco Gómez-Mula[1], Alejandro Cama-Pinto[3]

Resumen— **La agricultura de precisión y el Smart Farming son conceptos que están adquiriendo un importante auge por su relación con el internet de las cosas (IoT), especialmente en la búsqueda de nuevos mecanismos y procedimientos que permitan disponer de una agricultura sostenible y eficiente que posibilite satisfacer la demanda futura que requerirá la población a medida que aumenta. Ambos conceptos necesitan del despliegue de redes de sensores que monitoricen variables agrícolas para la integración de los datos de la agricultura espacial y temporal. En este trabajo se presenta el sistema que se ha desarrollado para medir la atenuación de las ondas en la banda libre (ICM) de 2.4 GHz cuando se propaga en el interior de un invernadero de tomate. Dicho sistema se basa en nodos Re-Mote del Zolertia con el sistema operativo Contiki y una Raspberry pi para el registro de los datos obtenidos. El nodo receptor registra el RSSI a distintas ubicaciones en el invernadero con el nodo transmisor y, además, a distintas alturas.**

Palabras clave—**Free Space Pathloss, Smart Farming, Internet de las cosas, Redes inalámbricas, WSN**

I. INTRODUCCIÓN

LOS conceptos tecnológicos de agricultura de precisión y Smart Farming están en auge por su relación con el Internet de las Cosas (IoT) [1], la agricultura sostenible, el manejo eficiente de los recursos y la producción mundial para satisfacer la demanda en alza de los alimentos, que según la FAO (Food and Agriculture Organization of the United Nations) predice que la población mundial aumentará de ocho mil millones de habitantes en el 2025 a nueve mil seiscientos millones en el 2050 [2]. En ese sentido, tanto la agricultura de precisión como Smart Farming necesitan del despliegue de redes de sensores que monitoricen variables agrícolas [3][4][5][6][7] para la integración de los datos de la agricultura espacial y temporal [8][9]. Estas redes se basan principalmente en protocolos de comunicaciones inalámbricas para facilitar el despliegue y evitar los cableados en un campo de cultivo [10][11][12]. Así mismo, suele utilizarse la banda de 2.4 GHz (2400 - 2483.5 MHz) por ser de uso libre a nivel mundial y el más difundido entre los módulos de radio de los fabricantes que emplean los estándares de comunicaciones inalámbricas WLAN [13][14][15], PAN como el IEEE 802.15.1 (Bluetooth), IEEE 802.15.4 empleado en WSN (Wireless Sensor Network) y IEEE 802.11 (WiFi).

[1] Department of Computer Architecture and Technology, University of Granada, 18071 Granada, Spain, e-mail: doracamapinto@correo.ugr.es, mdamas@ugr.es, frgomez@ugr.es.
[2] Software Engineering Department, University of Granada, 18071 Granada, Spain, e-mail: jholgado@ugr.es.
[3] Department of Computer Sciences and Electronic, Universidad de la Costa, Barranquilla 080002, Atlántico, Colombia, e-mail: acama1@cuc.edu.co.

En este trabajo estamos interesados en el desarrollo de un sistema de medición que facilite el registro del RSSI (Received Signal Strength Indicator) de las ondas de radio en la banda de 2.4 GHz que se propagan en cultivos en invernaderos, específicamente en invernaderos de tomates sobre los que vamos a realizar estudios concretos para validar el sistema desarrollado. En otro tipo de trabajos se han presentado diseños de sistemas similares al desarrollado que emplean el RSSI para encontrar el posicionamiento de sus nodos en el interior de un invernadero [16] [17][18].

En definitiva se presenta como se ha desarrollado el sistema de medición y registro de RSSI, así como su configuración, montaje y despliegue de la WSN (red inalámbrica de sensores) utilizando nodos Re-Mote de la empresa Zolertia para medir valores de RSSI a distintas distancias y alturas entre nodo Transmisor (Tx) y Receptor (Rx).

El artículo está dividido en varias secciones. En la sección II se describe la arquitectura, hardware y software del sistema de medida y registro desarrollado. En la sección III se detalla cómo se ha realizado el despliegue del sistema durante las pruebas de campo. En la sección IV se especifican los pasos seguidos para realizar las mediciones. En la sección V se discuten los resultados obtenidos, y en la sección VI las conclusiones de nuestro trabajo.

II. SISTEMA DE MEDICIÓN DE ATENUACIÓN DE ONDAS DE RADIO EN UN INVERNADERO

El sistema de medida desarrollado permite realizar mediciones del nivel de potencia de señal recepcionado (RSSI) en el interior de una explotación agrícola, y en particular en un invernadero. Hay otros estudios en esta temática, que también emplean redes de sensores inalámbricos - WSN (Wireless Sensor Network), y utilizan dispositivos modulares basados en placas Arduino y módulos de radio Xbee [19][20][21].

En nuestro caso, para la medición de RSSI de las ondas de radio utilizamos una placa que tiene integrado el módulo de radio en la banda ICM de 2.4 GHz [22]. Por otra parte, incluimos el registro de datos en el propio sistema para que sea autónomo, y pueda funcionar 24 horas durante dos semanas ininterrumpidamente. El principal beneficio del sistema desarrollado es su portabilidad, la facilidad de instalación en un entorno agrícola y un tiempo de autonomía suficiente para llevar a cabo una monitorización continua. El sistema está capacitado también para trabajar en la banda de 868 MHz, aunque en este estudio no se han realizado pruebas sobre dicha banda.

A. *Arquitectura*

La arquitectura del sistema se compone de dos estaciones, la estación receptora que registra el RSSI de la señal enviada desde la estación transmisora como se observa en las figuras 1 y 2. La estación receptora está compuesta por el nodo Re-Mote que se conecta y alimenta a través de su puerto USB conectado a un sistema empotrado Raspberry Pi, y este último a una toma de corriente de 220V que hay dentro del invernadero. El sistema es autónomo, pero para efectos de supervisión, se conecta esporádicamente a un ordenador portátil con un cable UTP al puerto RJ45 de la Raspberry Pi. Por su lado, la estación transmisora está conformada con el nodo Re-Mote que está alimentado por una batería recargable de litio-ion de 3.7 V con capacidad nominal de 6600 mAh que le otorga autonomía en su funcionamiento (figura 3Ay 3C). Estos elementos van dentro de una caja PVC con grado de protección IP65, preparado para no dejar entrar el polvo ni chorros de agua.

Fig. 1. Estación receptora que registra los valores de RSSI conectado a sistema empotrado Raspberry Pi.

La figura 2 muestra esquemáticamente la disposición de las dos estaciones en el cultivo. Cada estación está colocada sobre un mástil sostenida sobre una base de 17 kilogramos que le da mayor estabilidad. En los mástiles se agarran placas acrílicas que soportan las cajas PVC. La caja PVC en la estación transmisora alberga al nodo Re-Mote y una batería que la alimenta dándole autonomía. En el lado del receptor, la caja PVC tiene en su interior el nodo Re-Mote intercomunicado y alimentado por su conexión USB con la Raspberry Pi que es la unidad de registro de datos con su cargador enchufado a una toma corriente de 220V.

B. *Hardware del sistema*

El sistema de medición y registro está compuesto por los siguientes componentes hardware:

- **Motas Re-Mote**. Las motas Re-Mote de la empresa Zolertia tienen módulos de radio transceptores capaces de actuar como nodo transmisor Tx y receptor Rx. Es elegida porque su placa es una plataforma IoT con un amplio soporte de software de Contiki OS que incluye 6LoWPAN, RPL y otros protocolos IoT ampliamente utilizados. Integra el chip CC2538 System-on-Chip (SoC) de Texas Instruments para la comunicación de bajo consumo y corto alcance en la banda de 2.4 GHz, con un consumo de corriente de 24 mA cuando trasmite, 20 mA cuando recibe, y 1,3 uA en estado dormido [23][24][25]. Por otro lado, el PIRE (Potencia Isotrópica Radiada Equivalente) de nodo Tx fue de -29 dBm en las pruebas y se usaron para ambas motas antenas de 5 dBi de ganancia. La sensibilidad de recepción en los nodos Re-Mote es de -97 dBm.
- **Raspberry Pi.** Se ha seleccionado una Raspberry Pi como servidor que almacena la información de las mediciones en una memoria SD (figura 1) en formato CSV. Está conectado y alimentado a la mota Rx a través de su puerto USB [26].
- **Batería de Ion-Litio.** La batería de iones de litio se conecta a la mota transmisora Tx de 3.7 Voltios para mantener la autonomía del transmisor.
- **Sensor de humedad y temperatura.** En la mota transmisora se conecta adicionalmente el sensor DHT22 para trasmitir datos de temperatura y humedad, que tradicionalmente se utilizan para monitorizar y supervisar el estado ambiental del cultivo en un invernadero.

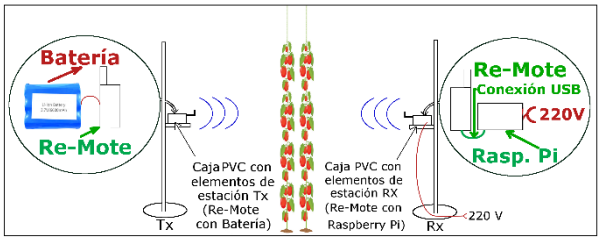


Fig. 2. Despliegue de las dos estaciones Tx y Rx comunicándose inalámbricamente en un corte transversal de un invernadero de tomate.

C. *Software del sistema*

Para el buen funcionamiento del sistema se requiere desarrollar la infraestructura software adecuada para minimizar fundamentalmente el consumo de energía. A continuación, se especifica el software desarrollado en cada caso.

1 **Motas Re-Mote**. Se ha instalado y configurado el sistema operativo Contiki, desarrollado en 2002 por Adam Dunkels, como entorno de ejecución de código abierto [27] para nodos de sensores inalámbricos de memoria limitada y de baja potencia. Es ligero por lo que es ideal para IoT. Sus aplicaciones se desarrollan con el lenguaje de programación C. Cuenta con una implementación TCP/IP incorporada para dispositivos integrados, siendo oficialmente compatible con diversas plataformas de dispositivos que conforman las redes de sensores inalámbricos, incluido la placa Re-Mote [28][29][3].

Se emplea solamente el módulo de ahorro de energía (power-mgmt.h) de Contiki en el nodo transmisor porque durante la etapa de prueba esta estación es la que se aleja del nodo receptor y no dispone de una toma de corriente eléctrica, sino que está alimentado por su batería. Por su parte, el nodo receptor se alimenta de la Raspberry Pi, que a su vez está conectada a una toma de corriente en un extremo del invernadero.

Para la comunicación de la radio empleamos la pila “Rime” (rime.h), que proporciona un conjunto de

primitivas de comunicación básica de difusión de red de un solo salto (“unicast”), del mejor esfuerzo, y el “unicast multisalto” confiable de múltiples saltos [30].

El programa desarrollado en C para la estación transmisora envía tramas de datos de temperatura y de humedad obtenidos del sensor DHT22 periódicamente con un temporizador variable, manteniéndose en suspensión el resto del tiempo. Esto permite reducir el consumo de energía, alargando la autonomía del transmisor. Por otra parte, la estación receptora alimentada con la Raspberry Pi y la toma de corriente del invernadero, mide el RSSI y obtiene la trama de datos enviada por el nodo transmisor.

- **Raspberry Pi**. Se ha instalado la distribución Raspbian basada en Debian y se han desarrollado varios scripts en lenguaje Python que establecen comunicación vía serial con los dispositivos Zolertia y generan archivos .csv con los datos que reciben, almacenándolos en la memoria SD de la Raspberry Pi. Tiene también un módulo de reloj con la pila CR2032 para que no se descalibre la fecha y hora cuando se apaga, grabándola con cada registro del RSSI.

III. DESPLIEGUE Y PUESTA EN MARCHA EN UN INVERNADERO.

Esta sección detalla cómo se ha realizado el despliegue del sistema y cómo se han diseñado los experimentos en un invernadero de tomates ubicado en la provincia de Almería, Comunidad Autónoma de Andalucía – España. Dicho producto, además de su valor nutricional, tiene una alta demanda en los mercados de la Comunidad Europea.

A. Despliegue del sistema

Los nodos Tx (Figura 3C) y Rx (Figura 3B) se ubican a diferentes distancias dentro del invernadero (Figura 3D) y a diferentes alturas con la ayuda de mástiles. La estación se coloca sobre una base de 17 kg de peso en el mástil para darles mayor estabilidad (Figura 3A). Ambas estaciones se instalan a las mismas alturas, y dicha altura se varía a lo largo de las mediciones. En general, las plantas de la tomatera se encuentran alineadas formando un muro y equidistantes entre ellas, separadas por un espacio o carril por donde transitan los agricultores, vista en la figura 3D.

B. Realización de experimentos

Para la realización de los experimentos se registraron valores cada 10 segundos durante 10 minutos en cada posición. La altura de los nodos era la misma para el nodo Tx y Rx en cada etapa de la medición. Además de las pruebas de registro y análisis de RSSI, también se pueden realizar funciones de monitorización de variables agrícolas y ambientales al enchufarles sensores analógicos y digitales en la mota Re-Mote, por ejemplo, de humedad o temperatura.

Fig. 3. A) Ubicación de nodo transmisor en el interior del invernadero, B) Nodo receptor conectado al computador embebido Raspberry Pi, C) Nodo transmisor alimentado por batería Litio-Ion de 3.7 V – 6600 mAh, D) Vista interna del invernadero de Tomate.

IV. MEDIDAS SOBRE UN INVERNADERO DE TOMATES.

Con el sistema propuesto se pueden realizar diferentes tipos de estudios para identificar los modelos de propagación de las ondas de radio en huertos [31], invernaderos de mango [32] y de tomates [22]. En nuestro caso seguimos los siguientes pasos:

- **Paso 1**. La posición de la estación receptora se mantiene siempre fija, ya que se encuentra alimentada con una corriente eléctrica de 220 V en una toma de corriente del invernadero. En la figura 4 puede observarse las posiciones de los nodos receptores representados en color rojo en un extremo del invernadero.
- **Paso 2**. De manera similar la estación transmisora representada con Tx en la figura 5 y con color celeste en la figura 4 se irá alejando del nodo Rx. Primero el nodo Tx se coloca en la posición B1 y el nodo Rx en la posición A1, distanciados 2,6 metros (ver figura 5) y se realiza una medición de RSSI. Posteriormente ambas estaciones se mueven hacia la derecha (A2 y B2) y se realiza otra medición, y así sucesivamente hasta llegar a las ubicaciones A4 y B4 (ver figura 4). La distancia de 2,6 metros depende de la hilera de plantas y la separación que hay entre los carriles en el invernadero; en este caso, solo hay una hilera de plantas de separación entre ellas.
- **Paso 3**. Las dos estaciones se alejan entre ellas a una distancia inicial de 4,4 metros (ver figura 5), que corresponde con la distancia de dos hileras de plantas entre la estación transmisora y receptora. La estación Tx se coloca en las posiciones C1,C2,C3,C4, y se va registrando el RSSI con la estación Rx en las posiciones A1,A2,A3,A4 respectivamente.
- **Paso 4**. Ahora las dos estaciones se alejan entre ellas a una distancia inicial de 6,2 metros (ver figura 5), correspondiente a la distancia de tres hileras de plantas. La estación Tx se va colocando en las posiciones C1,C2,C3,C4, y registrando el RSSI con la estación Rx en las posiciones A1,A2,A3,A4 respectivamente.
- **Paso 5**. Después que la estación Tx llega al extremo opuesto del invernadero, se repite el procedimiento cambiando a una nueva altura ambos nodos, por

ejemplo, a 0.5 metros sobre el suelo. Luego se repiten los pasos empezando por el paso 1.

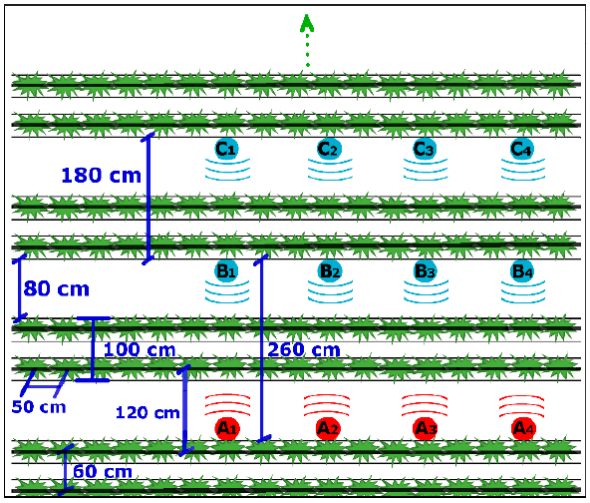


Fig. 4. Detalles del marco de plantación dentro del invernadero y posiciones cambiantes del nodo transmisor (color celeste) y receptor (rojo).

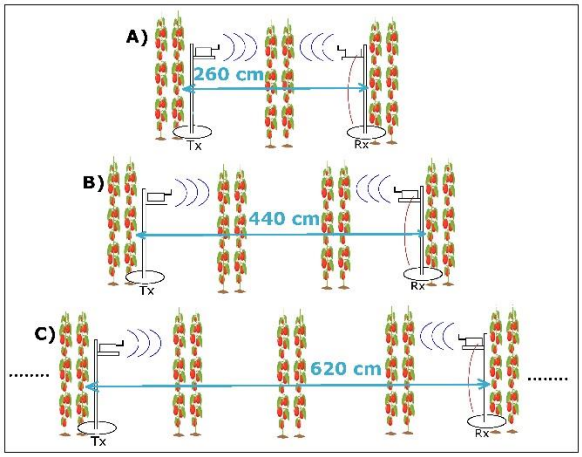


Fig. 5. Diagrama transversal de comunicación inalámbrica entre estación Tx y Rx en invernadero de tomates

V. DISCUSIÓN

A partir de las medidas realizadas a diferentes distancias y alturas podemos analizar el valor promedio de las mediciones de RSSI y estudiar cómo las señales de ondas de radio en 2400 MHz se atenúan a medida que atraviesan las hileras de plantas de tomateras (figura 6).

En la figura 6, observamos que a 50 cm sobre el suelo se alcanza la máxima distancia entre la comunicación de los dos nodos Tx y Rx que es 24.2 metros, y la mínima distancia de cobertura entre ellos (13.4 metros) cuando los nodos son colocados a 1.5 metros del suelo. Los valores fueron registrándose en el nodo receptor hasta que se acercaban a los -100 dBm porque la sensibilidad del receptor es de -97 dBm. Sin embargo, para efectos de estabilidad en el enlace inalámbrico, se sugiere tener un margen entre la potencia que detecta el receptor y la sensibilidad de recepción igual o mayor a 10 dB [33].

Por otro lado, nuestro sistema basado en nodos Re-Mote, es más compacto que otros sistemas de mayor tamaño y con un transceptor no integrado, y al aprender a configurarlas con el sistema operativo Contiki, se adquieren conocimientos y destrezas en su programación para futuros proyectos de monitorización ligados a IoT (Smart Agriculture) y a la agricultura de precisión. Por otra parte el sistema desarrollado tiene un precio unitario aproximadamente en 130 euros, muy por debajo de otras opciones del mercado.

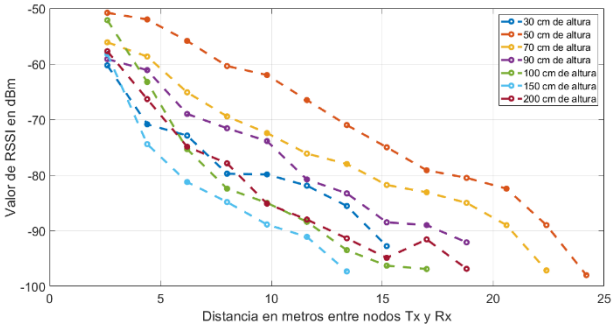


Fig. 6. Niveles de potencia de la señal en dBm entre el nodo transmisor (Tx) y receptor (Rx) a distintas alturas y distancias entre ellos [7].

La figura 7 muestra las curvas de atenuación de la onda de radio entre el nodo Tx y Rx a 0.5 y 1.5 metros del suelo. Por ejemplo, el valor promedio de RSSI a 2.6 metros de distancia entre ambos nodos y a 150 cm del suelo es -58.22 dBm (Ver figura 6). Así, la atenuación en la trayectoria se obtiene con el siguiente cálculo:

$$\text{PIRE} + \text{Atenuación}_{\text{trayecto}} + \text{Ganancia}_{\text{RX}} = -58.22 \text{ dBm}$$

Por lo tanto,

$$\text{Atenuación}_{\text{trayecto}} = -58.22 \text{ dBm} + 29\text{dBm} - 5\text{dBi}$$
$$\text{Atenuación}_{\text{trayecto}} = -34.22 \text{ dB}$$

La atenuación producida durante el trayecto de la onda de radio entre el nodo Tx y Rx es de -34.22 dB a una altura de 1.5 metros sobre el suelo (figura 7). Las dos curvas mostradas en la figura 7 son importantes porque resumen el mayor y nivel de la atenuación por la presencia vegetación, y por tanto de mayor y menor cobertura cuando los nodos están dispuestos a 0.5 m y 1.5m del suelo respectivamente.

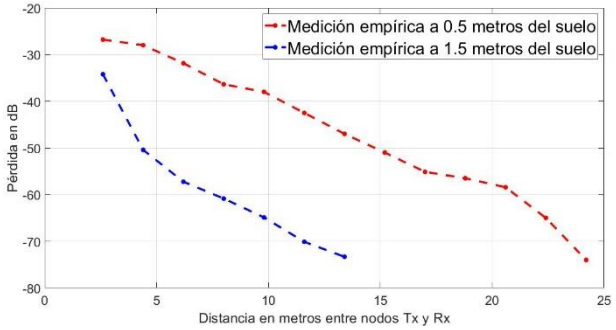


Fig. 7. Pérdida o atenuación de trayecto medida en dB de la onda de radio entre el nodo Tx y Rx al interior del invernadero de tomate a 0.5 y 1.5 metros del suelo

VI. CONCLUSIONES

Los resultados obtenidos contribuyen a planificar mejor la cobertura que hay en la banda de 2.4 GHz en el interior de un invernadero para establecer el número de nodos necesarios en el despliegue de motas utilizadas para la monitorización, determinando la mejor ubicación y altura de los nodos Tx y Rx.

En este escenario particular las medidas obtenidas nos permite determinar la altura de los nodos respecto al suelo más adecuada para ofrecer una mayor cobertura o distancia en el enlace inalámbrico de un solo salto, consiguiéndolo cuando se sitúan los nodos Tx y Rx a 0.5 metros sobre el suelo, mientras que la menor cobertura se alcanza cuando los nodos se colocan a 1.5 metros del suelo.

Además, nuestro sistema demostró ser eficiente durante la etapa de pruebas de campo, y se prevé como trabajo futuro usarlo para la medición y registro de RSSI en otros invernaderos de distintos cultivos para comprender mejor las curvas de atenuación que provocan sobre las ondas de radio en la banda de 2400 MHz. De igual modo se determinará la atenuación en el caso que se utilice la frecuencia libre como 868 MHz en lugar de 2400 MHz.

REFERENCIAS

[1] Rodríguez-Valenzuela, S., Holgado-Terriza, J., Gutiérrez-Guerrero, J., & Muros-Cobos, J. Distributed service-based approach for sensor data fusion in IoT environments. Sensors, 14(10), 19200-19228, 2014.

[2] Razafimandimby, C.; Loscrí, V.; Vegni, A.M.; Neri, A. *Efficient Bayesian communication approach for smart agriculture applications.* In Proceedings of the 2017 IEEE Vehicular Technology Conference, Toronto, ON Canada, 24–27 September 2017; pp. 1–5, 2017.

[3] Caicedo-Ortiz, J.G., De-la-Hoz-Franco, E., Morales Ortega, R., Piñeres-Espitia, G., Combita-Niño, H., Estévez, F., Cama-Pinto, A. *Monitoring system for agronomic variables based in WSN technology on cassava crops*. Computers and Electronics in Agriculture, 145, pp. 275-281. DOI: 10.1016/j.compag.2018.01.004, 2018.

[4] Cama-Pinto, A., Gil-Montoya, F., Gómez-López, J., García-Cruz, A., Manzano-Agugliaro, F. *Wireless surveillance sytem for greenhouse crops*. DYNA (Colombia), 81 (184), pp. 164-170. DOI: 10.15446/dyna.v81n184.37034, 2014.

[5] Cama, A., Montoya, F.G., Gómez, J., De La Cruz, J.L., ManzanoAgugliaro, F. *Integration of communication technologies in sensor networks to monitor the Amazon environment*. Journal of Cleaner Production, 59, pp. 32-42.DOI:10.1016/j.jclepro.2013.06.041, 2013.

[6] Montoya, F.G., Gomez, J., Manzano-Agugliaro, F., Cama, A., García-Cruz, A., De La Cruz, J.L. *6LoWSoft: A software suite for the design of outdoor environmental measurements*. Journal of Food, Agriculture and Environment, 11 (3-4), pp. 2584-2586, 2013.

[7] Montoya, F.G., Gómez, J., Cama, A., Zapata-Sierra, A., Martínez, F., De La Cruz, J.L., Manzano-Agugliaro, F. *A monitoring system for intensive agriculture based on mesh networks and the android system.* Computers and Electronics in Agriculture, 99, pp. 14-20. DOI: 10.1016/j.compag.2013.08.028, 2013.

[8] Gómez, J., Montoya, F. G., Manzano-Agugliaro, F., Cama, A. *Sistema de monitorización a través de 6lowpan para la recolección de variables aplicadas a la agricultura de precisión.* III Jornadas de Computación Empotrada (JCE), 19-21, 2012.

[9] Zapata-Sierra, A.J., Cama-Pinto, A., Montoya, F.G., Alcayde, A., Manzano-Agugliaro, F. *Wind missing data arrangement using wavelet based techniques for getting maximum likelihood*. Energy Conversion and Management, 185, pp. 552-561. DOI: 10.1016/j.enconman.2019.01.109, 2019.

[10] Cama-Pinto, A., Piñeres-Espitia, G., Comas-González, Z., VélezZapata, J., Gómez-Mula, F. *Design of a monitoring network of meteorological variables related to tornadoes in BarranquillaColombia and its metropolitan area*. Ingeniare, 25 (4), pp. 585-598. DOI: 10.4067/S0718-33052017000400585, 2017.

[11] Cama-Pinto, A., Piñeres-Espitia, G., Zamora-Musa, R., Acosta-Coll, M., Caicedo-Ortiz, J., Sepúlveda-Ojeda, J. *Design of a wireless sensor network for monitoring of flash floods in the city of Barranquilla, Colombia*. Ingeniare, 24 (4), pp. 581-599, 2016.

[12] Caicedo Ortiz, J. G., Acosta Coll, M. A., & Cama-Pinto, A. WSN deployment model for measuring climate variables that cause strong precipitation. Prospectiva, 13(1), 106-115, 2015.

[13] Piñeres-Espitia, G., Cama-Pinto, A., De La Rosa Morrón, D., Estevez, F., Cama-Pinto, D. *Design of a low cost weather station for detecting environmental changes*. Espacios, 38 (59), 13, 2017.

[14] Foerster, A., Udugama, A., Görg, C., Kuladinithi, K., Timm-Giel, A., Cama-Pinto, A. *A novel data dissemination model for organic data flows*. Lecture Notes of the Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering, LNICST, 158, pp. 239-252. DOI: 10.1007/978-3-319-26925-2_18, 2015.

[15] Cama, A., De la Hoz, E., Cama, D. *Las redes de sensores inalámbricos y el internet de las cosas*. Revista INGE CUC, 8(1), pp. 163-172 pp. 163-172, 2012.

[16] Subashini, M.M., Das, S., Heble, S., Raj, U., Karthik, R. *Internet of things based wireless plant sensor for smart farming*. Indonesian Journal of Electrical Engineering and Computer Science, 10 (2), pp. 456-468. DOI: 10.11591/ijeecs.v10.i2.pp456-468, 2018.

[17] Xu, L. *Design of a RSSI Location System for Greenhouse Environment*. International Journal of Distributed Sensor Networks, 2015, art. no. 525861. DOI: 10.1155/2015/525861, 2015.

[18] Abouzar, P., Michelson, D.G., Hamdi, M. *RSSI-Based Distributed Self-Localization for Wireless Sensor Networks Used in Precision Agriculture*. IEEE Transactions on Wireless Communications, 15 (10), art. no. 7505647, pp. 6638-6650. DOI: 10.1109/TWC.2016.2586844, 2016.

[19] Widodo, S., Pratama, E.A., Pramono, S., Budi Basuki, S. Outdoor propagation modeling for wireless sensor networks 2.4 GHz. IEEE International Conference on Communication, Networks and Satellite, COMNETSAT 2017 - Proceedings, 2018-January, pp. 158-162. DOI: 10.1109/COMNETSAT.2017.8263592, 2018.

[20] Cama-Pinto, A., Piñeres-Espitia, G., Caicedo-Ortiz, J., Ramírez-Cerpa, E., Betancur-Agudelo, L., Gómez-Mula, F. Received strength signal intensity performance analysis in wireless sensor network using Arduino platform and XBee wireless modules. International Journal of Distributed Sensor Networks, 13 (7), DOI: 10.1177/1550147717722691, 2017.

[21] Shue, S., Johnson, L.E., Conrad, J.M. Utilization of XBee ZigBee modules and MATLAB for RSSI localization applications, Conference Proceedings - IEEE SOUTHEASTCON, art. no. 7925305, DOI: 10.1109/SECON.2017.7925305, 2017.

[22] Cama-Pinto, D., Damas, M., Holgado-Terriza, J. A., Gómez-Mula, F., & Cama-Pinto, A. Path Loss Determination Using Linear and Cubic Regression Inside a Classic Tomato Greenhouse. International journal of environmental research and public health, 16(10), 1744, 2019.

[23] B. Van Herbruggen, B. Jooris, J. Rossey, M. Ridolfi, N. Macoir, Q. Van Den Brande, S. Lemey, E. De Poorter. *Wi-pos: A low-cost, open source ultra-wideband (UWB) hardware platform with long range sub-GHZ backbone*. Sensors (Switzerland), 19 (7), art. no. 1548, DOI: 10.3390/s19071548, 2019.

[24] Bezunartea, M., Wang, C., Braeken, A., Steenhaut, K. *Multi-radio Solution for Improving Reliability in RPL*. IEEE International Symposium on Personal, Indoor and Mobile Radio Communications, PIMRC, 2018-September, art. no. 8580913, pp. 129-134. DOI: 10.1109/PIMRC.2018.8580913, 2018.

[25] Texas Instruments–Descripción CC2538. Visitado el 02/06/2019 en: http://www.ti.com/product/CC2538/description

[26] Gomez, J., Villar, E., Molero, G., Cama, A. Evaluation of high performance clusters in private cloud computing environments. In Distributed Computing and Artificial Intelligence (pp. 305-312). Springer, Berlin, Heidelberg, 2012.

[27] ERCIM News. Contiki: Bringing IP to Sensor Networks. Visitado el 08/06/2019 en: https://ercim-news.ercim.eu/en76/rd/contiki-bringing-ip-to-sensor-networks.

[28] Dunkels, A., Grönvall, B., Voigt, T. Contiki - A lightweight and flexible operating system for tiny networked sensors. Proceedings

- Conference on Local Computer Networks, LCN, pp. 455-462. DOI: 10.1109/LCN.2004.38, 2004.

[29] Staudemeyer, R.C., Pöhls, H.C., Wójcik, M. *What it takes to boost Internet of Things privacy beyond encryption with unobservable communication: a survey and lessons learned from the first implementation of DC-net*. Journal of Reliable Intelligent Environments, 5 (1), pp. 41-64. DOI: 10.1007/s40860-019-00075-0, 2019.

[30] Dunkels, A., Österlind, F., He, Z. *An adaptive communication architecture for wireless sensor networks*. SenSys'07 - Proceedings of the 5th ACM Conference on Embedded Networked Sensor Systems, pp. 335-349. DOI: 10.1145/1322263.1322295, 2007.

[31] Vougioukas, S.; Anastassiu, H.T.; Regen, C.; Zude, M. *Influence of foliage on radio path losses (PLs) for Wireless Sensor Network (WSN) planning in orchards*. Biosyst. Eng., 114, 454–465, 2013.

[32] Raheemah, A.; Sabri, N.; Salim, M.S.; Ehkan, P.; Ahmad, R.B. *New empirical path loss model for wireless sensor networks in mango greenhouses*. Comput. Electron. Agric., 127, 553–560, 2016.

[33] Zennaro, M.; Bagula, A.; Gascon, D.; Noveleta, A.B. *Long distance wireless sensor networks: Simulation vs. reality*. In Proceedings of the 4th ACM Workshop on Networked Systems for Developing Regions, NSDR '10, San Francisco, CA, USA, 15 June 2010.

Construyendo un Dispositivo de Internet de las Cosas para el Hogar Conectado

Driss Iounes[1], Juan Manuel López-Torralba[1], Pablo Pico-Valencia[2], Juan Antonio Holgado-Terriza[1]

***Resumen*— El internet de las cosas (IoT, Internet of Things) está propiciando una creciente demanda de dispositivos u objetos IoT en el Hogar Digital Conectado para mejorar aspectos como la gestión de la eficiencia energética, el confort, la seguridad perimetral, los sistemas de cuidado o teleasistencia, y los servicios multimedia avanzados, entre otros. Sin embargo, la gran diversidad de alternativas, la diferencia entre tipos de redes inalámbricas, la falta de seguridad y la privacidad de los datos o la dificultad en su puesta en marcha complican el diseño y construcción de objetos IoT que encajen en las plataformas IoT existentes en el mercado. En este trabajo, analizamos cuáles son los aspectos necesarios para llevar a cabo la construcción y desarrollo de un objeto IoT que tiene que garantizar, además de la fiabilidad, escalabilidad y flexibilidad en su configuración y despliegue, la seguridad en cuanto a la transmisión de los datos y la confiabilidad en cuanto a la protección de los datos personales manejados. Se explicarán los pasos que se ha seguido en la construcción de un objeto IoT sencillo, consistente en una cerradura inteligente (smart lock), y su posterior integración en un ecosistema IoT.**

***Palabras clave*— Hogar conectado, IoT, MQTT.**

I. INTRODUCCIÓN

Los avances tecnológicos que se están produciendo en productos de electrónica de consumo están propiciando una creciente demanda de dispositivos u objetos del internet de las cosas (IoT, Internet of Things), en especial en el campo del hogar digital conectado. En general, el mercado ofrece sensores y actuadores inalámbricos cada vez más específicos (bombillas, enchufes, termostatos, altavoces) integrados en microcontroladores o sistemas empotrados de recursos limitados relativamente potentes y de bajo consumo de energía, y que al mismo tiempo ofrecen una conectividad IP con protocolos TCP a Internet [1][2].

En muchos casos dichos objetos conectados suelen también ser inteligentes basándose en la capacidad de interconexión y colaboración que tienen con otros dispositivos o sistemas que se encuentran disponibles a través de Internet, lo que les permite tomar decisiones adaptadas al entorno. Por ejemplo, un termostato inteligente puede decidir la acción de control sobre una caldera en base a la consigna de temperatura establecida por el usuario y el estado ambiental del entorno en que se encuentra ubicado mediante la conexión con un sistema meteorológico que proporcione una predicción del tiempo [3][4][5].

[1] Universidad de Granada, Granada, España, e-mail: diounes@correo.ugr.es, ltjuanma94@ugr.es, jholgado@ugr.es.

[2] Pontificia Universidad Católica del Ecuador, Esmeraldas, Ecuador, e-mail: pablo.pico@pucese.edu.ec.

En este ambiente de interacción del Hogar Digital Conectado no sólo se producen interacciones entre objetos IoT sino también entre servicios, aplicaciones, plataformas, redes de comunicaciones y humanos, estos últimos a través de sus dispositivos móviles, wearables o táctiles. Todas estas interacciones conforman el ecosistema del IoT [6]. Dentro de este contexto que viene a denominarse del Internet del Todo (IoE, Internet of Everything) [7] y que será la base del Futuro Internet [8] será posible desplegar los agentes o aplicaciones autónomas capaces de llevar a cabo comportamientos inteligentes y proactivos mediante la exploración, el conocimiento, el procesamiento y análisis de los datos que obtiene en sus interacciones o colaboraciones con el resto del ecosistema IoT [9].

Existen diversas propuestas de arquitecturas de referencia para ecosistemas de IoT [10][11] que se sustentan generalmente en modelo de capas. Así, en la propuesta de Da Xu et al. [10] y Chen et al. [11] se define un modelo de cuatro capas que incluyen una capa de percepción ligada a los objetos IoT capaces de explorar o actuar sobre el entorno, una capa de red sobre la que interoperan los distintos objetos IoT, una capa de middleware en el que se implementan los mecanismos de interacción, gestión y persistencia de los información manejada en el ecosistema a través de la nube y, por último, la capa de aplicación que se encarga del procesamiento de dicha información para el usuario final [12].

Aunque en los ecosistemas IoT para el Hogar Digital Conectado los objetos IoT utilizan diferentes tipos de redes de comunicaciones, generalmente inalámbricos (ZigBee, Bluetooth, Z-Wave) para conectarse a la plataforma IoT, es común establecer una conectividad IP punto a punto para su exposición a Internet para así poder ser manejado desde otras aplicaciones, servicios o terminales móviles sin necesidad de utilizar pasarelas y repetidores para interactuar con dicho objeto. En muchos casos, las conectividades basadas en IP no siempre son sencillas al suponer un coste superior del dispositivo, un mayor consumo energético y un mayor coste en la conectividad que poco a poco se va reduciendo.

Por otra parte, la flexibilidad que supone el acceso desde cualquier parte del mundo también lleva aparejado consigo problemas como la falta de seguridad o la falta de privacidad en los datos manejados por dichos ecosistemas. Estas son cuestiones importantes si tenemos en cuenta que la capacidad inteligente y proactiva de los sistemas de IoT depende de un análisis detallado de los patrones de comportamiento de los habitantes de una casa. Se requiere por tanto, diseñar objetos IoT que aborden la seguridad y

la privacidad como aspectos prioritarios del propio objeto [13][14].

En este trabajo se aborda la problemática de la creación de un objeto IoT que garantice las propiedades de fiabilidad, escalabilidad y flexibilidad en su configuración y despliegue, y tenga presente la seguridad en cuanto a la transmisión y la protección de datos como aspectos fundamentales del mismo. Para ello, se analiza cómo debe ser el proceso de construcción de un objeto IoT y su integración en los ecosistemas IoT presentes en Hogar Digital Conectado.

El artículo está dividido en varias secciones. En la sección II se describe el proceso de creación y construcción de un objeto IoT y su posterior integración en un ecosistema IoT. En la sección III se detalla aspectos relacionados a la conectividad de un objeto inteligente dentro del Hogar Digital y los distintos protocolos de comunicación inalámbrica. En la sección IV se discute sobre el protocolo de conectividad MQTT (*Message Queue Telemetry Transport*). En la sección V se describen aspectos sobre la arquitectura IoT más adecuada para el Hogar Digital. En la sección VI se detallan los procedimientos a seguir para la construcción de un objeto IoT, en concreto la construcción de una cerradura inteligente. En la sección VII se especifican los pasos a seguir para la integración del dispositivo IoT en el ecosistema IoT. La sección VIII especifica cómo hacer la instalación del objeto en un entorno real. Finalmente, en la sección IX se presentan las conclusiones del trabajo.

II. PROCESO DE DESARROLLO DE UN OBJETO IOT

El proceso de creación y construcción de un objeto IoT requiere combinar aspectos como el diseño electrónico, el diseño mecánico, la fabricación del hardware, el desarrollo del programa controlador, la conectividad con el hogar digital, y su integración en un ecosistema de IoT, además de valorar la concepción de la idea del objeto IoT y las oportunidades de negocio que ofrece como cualquier otro producto de electrónica de consumo [15].

Existen varias alternativas para determinar el proceso de creación y desarrollo del objeto IoT que dependen esencialmente del procedimiento que se utiliza para integrar el objeto IoT en el ecosistema IoT [16]. Fundamentalmente se pueden distinguir dos etapas esenciales: el desarrollo y construcción del propio objeto IoT y su integración en un ecosistema IoT, tal y como se muestra en **Error! Reference source not found.**. En ambos casos, cada etapa está compuesta por un conjunto de subetapas consecutivas.

La construcción del objeto IoT implica realizar una serie de pasos que parten de la descripción de las funcionalidades esperables en el nuevo objeto IoT, el diseño electrónico (elección de PCB (*Printed Circuit Board*), componentes pasivos), el diseño mecánico que se requiere para el encapsulado de sensores o actuadores o la mecánica del propio objeto, la integración de los distintos elementos hardware (microcontrolador, sensores, actuadores) teniendo en cuenta el diseño electrónico que se ha llevado a cabo y, por último la programación del software en un firmware que puede actualizarse en el objeto IoT.

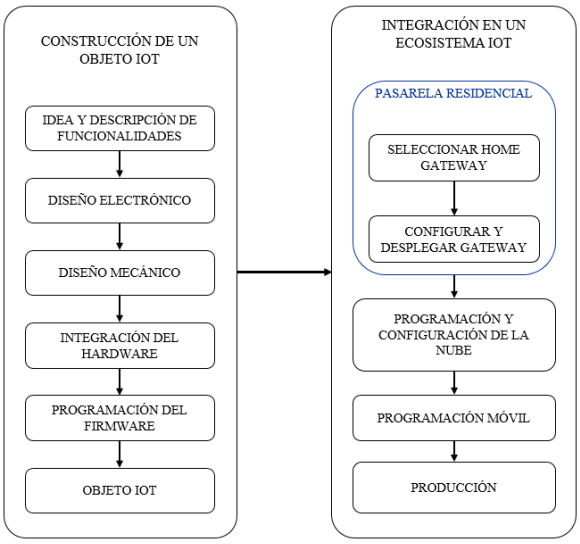


Fig. 1. Etapas seguidas para la construcción de un objeto IoT y su integración en un ecosistema IoT.

Una vez desarrollado el objeto IoT es importante estudiar cómo se va a integrar el producto final con el ecosistema IoT que se utilice de base. Esto requiere determinar los elementos adicionales que se deben considerar para que el objeto IoT pueda exponer públicamente su funcionalidad en Internet y, por tanto, sea accesible desde cualquier cliente que se ejecute en un ordenador o un dispositivo móvil. Las distintas subetapas se verán con más detalle cuando se defina más adelante la arquitectura de referencia.

III. LA CONECTIVIDAD EN HOGAR DIGITAL

La capacidad de conectividad de un objeto IoT es fundamental, ya que el objeto IoT debe estar expuesto a los posibles accesos que se puedan producir por parte de aplicaciones o servicios consumidores del objeto IoT desde cualquier ubicación del mundo y en cualquier instante de tiempo.

Dentro del hogar se utilizan diferentes tipos de protocolos de comunicación, principalmente inalámbricos, en base a la necesidad de autonomía, el consumo energético, el alcance, la frecuencia y el caudal de transmisión. En la Tabla I se muestra un resumen de las principales características asociadas a estos estándares de comunicación, entre las que se incluyen el alcance y la posibilidad de interoperar entre varias redes IP.

Los protocolos de comunicación basados en TCP/IP utilizan el mismo protocolo tanto para las redes privadas, locales y anchas. Esto simplifica el acceso desde redes de banda ancha como desde redes de comunicación móvil como 4G. En la Tabla II se muestran los protocolos de aplicación sobre TCP/IP más comunes definidos para la implementación de los servicios de Internet. Dentro de esta lista se puede distinguir protocolos basados en dos paradigmas de comunicación, esto es, paradigma de tipo petición/respuesta y los paradigmas de publicación/suscripción.

El paradigma petición/respuesta es el más utilizado y en particular usa el protocolo HTTP (*Hypertext Transfer Protocol*). Sin embargo, los protocolos basados en publicación/suscripción pueden reducir el intercambio de

TABLA I

COMPARACIÓN DE PROTOCOLOS INALÁMBRICOS EXISTENTES EN IOT PARA HOGAR DIGITAL CONECTADO

Protocolo	**Caudal**	**Banda de frecuencia**	**Alcance**	**Interoperabilidad con IP**
EnOcean	125 kbps	868 MHz	30 m (En línea recta)	Necesita puente EnOcean.
Z-Wave	50 kbps	868 MHz	1-40 m (En línea recta)	Necesita puente Z-Wave.
ZigBee	20 kbps	868 MHz, 2.4 GHz	10 m (En línea recta)	Necesita puente ZigBee.
6LoWPAN	250 kbps	915 MHz, 868 MHz	20 m	Sí
Bluetooth	1 Mbps	2402 – 2480 MHz	10 m (En línea recta)	Necesita puente
BLE	2 Mbps	2.4 GHz	70 m (En línea recta)	Necesita puente
802.11 a	25 Mbps	5.15 – 5.35 GHz, 5.47 GHz	25 m	Sí
802.11 b	6.5 Mbps	2.4, 5.0 GHz	35 m	Sí
802.11 g	25 Mbps	2.4, 5.0 GHz	25 m	Sí
802.11 n	200 Mbps	2.4, 5.0 GHz	50 m	Sí
802.11 ah	0.6 – 8.0 Mbps	900 MHz	100 m (obstáculos incluidos)	Sí

mensajes especialmente cuando se envían paquetes pequeños como ocurre en la transmisión de los datos de un sensor o la actuación de un sensor. En este sentido los protocolos CoAP (*Constrained Application Protocol*) y MQTT son buenos candidatos para Hogar Digital por ser protocolos muy ligeros comparados con el clásico HTTP. MQTT en particular garantiza fiabilidad en la comunicación por ser orientado a conexión y utiliza un paradigma publicación/suscripción, lo que permite utilizar una arquitectura basada en eventos en lugar de una arquitectura cliente/servidor basada en petición/respuesta. Por otra parte, MQTT tiene soporte a la seguridad que se basa en el uso de TLS (*Transport Layer Security*) sobre el canal TCP.

IV. MQTT

MQTT es un protocolo de conectividad M2M (*Machine to Machine*) de código abierto que fue diseñado por IBM como un protocolo de transporte de mensajería de publicación/suscripción extremadamente ligera. El sistema se compone de dos entidades: los clientes que pueden ser publicadores o suscriptores, y el bróker o servidor que se encarga de distribuir los mensajes entre publicadores y suscriptores. Los suscriptores se registran en el bróker especificando los tópicos o temas en los que están interesados. Cuando los publicadores envían mensajes al bróker sobre tópicos concretos, el bróker se encarga de reenviar los mensajes a todos los clientes suscritos sobre dicho tema. En Fig. 2 se muestra un ejemplo claro de suscriptores sobre dos tópicos en el bróker y los clientes suscriptores.

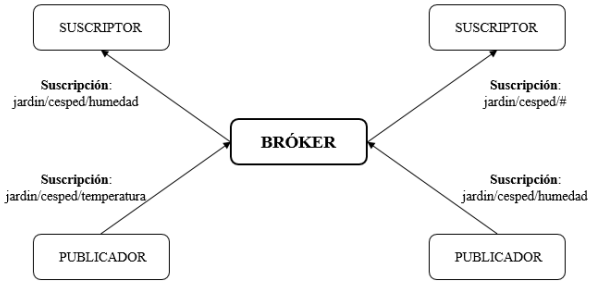


Fig. 2. Arquitectura básica de MQTT.

El tópico consta de uno o más niveles temáticos. Cada nivel de tópico está separado por una barra diagonal que sirve como separador de nivel de tópico. Entonces, los tópicos de MQTT están estructurados en una jerarquía similar a los directorios en un sistema de archivos, es decir, utilizan la barra (/) como un delimitador. Por ejemplo: Temperatura/ o Temperatura/dormitorio.

El protocolo MQTT se diferencia del protocolo HTTP en tener tres niveles de calidad de servicio (QoS): 0 = a lo sumo una vez (mejor esfuerzo, no ack); 1 = al menos una vez (retransmitido si no se recibe ack); y 2 = exactamente una vez.

Este protocolo incluye también el concepto de mensajes retenidos que puede ser de utilidad. El servidor mantiene los mensajes incluso después de enviarlos a todos los suscriptores, por lo que los nuevos suscriptores pueden recibir los mensajes retenidos. Los mensajes más utilizados son generalmente CONNECT, PUBLISH, SUBSCRIBE Y UNSUBSCRIBE.

Con respecto a la seguridad, MQTT incluye control de acceso al bróker o a un tópico mediante autenticación usuario/contraseña, aunque los mensajes son transmitidos en texto plano. También puede utilizarse el protocolo TLS

TABLA II

PROTOCOLOS DE LA CAPA DE APLICACIÓN

Protocolos	**Capas de transporte**	**Paradigma de comunicación**	**Seguridad y QoS (Quality of Service)**	**Tamaño de encabezado (bytes)**	**Longitud máxima (bytes)**
HTTP	TCP	Solicitud/Respuesta	Ambos	-	-
UPnP	TCP	Publicación/Suscripción	Ambos	-	-
MQTT	TCP (larga duración)	Publicación/Suscripción	Ambos	2	5
XMPP	TCP	Ambos	Seguridad	-	-
AMQP	TCP	Publicación/Suscripción	Ambos	8	-
CoAP	UDP	Solicitud/Respuesta	Ambos	4	20

sobre TCP/IP para proporcionar un canal seguro cifrado para la transmisión de mensajes MQTT. Esto protege todas las partes del mensaje MQTT, y no sólo el *payload* del mensaje. Actualmente la versión más estable es el TLS v1.2.

Las restricciones de los tópicos se configuran en el bróker y no es necesario configurar nada en el cliente. La restricción de los tópicos se realiza en un archivo de lista de control de acceso (ACL).

V. ECOSISTEMA IOT EN HOGAR DIGITAL

El ecosistema IoT determina cómo se van a producir las interacciones con el objeto IoT y con otras entidades distribuidas en la red tales como las aplicaciones, servicios, agentes, e incluso otros humanos a través de dispositivos móviles. La arquitectura más general del ecosistema IoT para Hogar Digital, ilustrada en Fig. 3, se compone de tres niveles: nivel de percepción, nivel de pasarela local o remota y nivel de clientes consumidores.

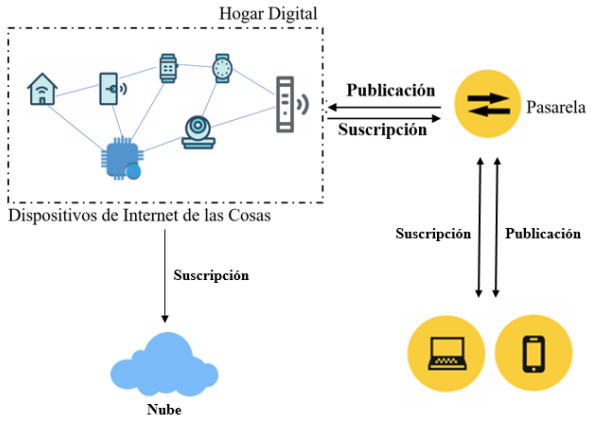


Fig. 3. Arquitectura genérica con 3 niveles para el Hogar Digital Conectado.

En el nivel de percepción se incluye la interconexión de todos los objetos IoT en una red local privada. El nivel de pasarela residencial (*Home Gateway*) se encarga de coordinar las acciones/consultas hacia/con los objetos IoT en la red local de la casa, así como recoger/enviar peticiones que llegan de Internet. A este nivel también se gestiona la persistencia de los datos y la gestión de su estado haciendo uso de sistemas basados en la nube o gestores de bases de datos. Por último, en el nivel de clientes se consideran los distintos medios que los usuarios usan para tener acceso a la red local del sistema.

VI. CONSTRUCCIÓN DE UN OBJETO IOT: CERRADURA INTELIGENTE

A. *Descripción de la cerradura inteligente*

Como caso de estudio se ha construido y desarrollado un objeto IoT sencillo, denominado *Smart Lock*. Su función principal consiste en gestionar la apertura y cierre automática de puertas en base a las necesidades de los clientes de la casa.

El sistema fue diseñado para abrir la puerta de forma remota con un dispositivo móvil y permitir el acceso sólo a los habitantes de la casa. Este dispositivo puede colaborar con cámaras inteligentes que pueden ser capaces de realizar reconocimiento facial para permitir o denegar el acceso a las personas según si están o no registradas en la base de datos de acceso.

B. *Diseño electrónico*

En este caso, para mover la cerradura fue necesario un motor con potencia suficiente para mover el perno de la puerta. Para ello, se ha utilizado un motor de corriente continua que puede girar en sentido horario y antihorario hasta que el microcontrolador corte la corriente del motor a través de un retardo de tiempo preconfigurado.

Para realizar el control del motor se ha utilizado un puente H que controla la velocidad y dirección de dos motores de corriente continua o un motor paso a paso de una forma muy sencilla. Dicho puente es un componente formado por 4 transistores que permite invertir el sentido de la corriente y así, invertir el sentido de giro del motor.

En el caso particular de este trabajo no se ha realizado el diseño de ninguna PCB (*Printed Circuit Board*) debido a que se aprovecha la integración de varias placas que tienen tanto el microcontrolador como el *driver* de corriente para controlar el motor que abre o cierra la puerta.

C. *Diseño mecánico*

El diseño mecánico de la cerradura inteligente se muestra en Fig. 4. Éste consiste, por un lado, en la soldadura de uno de los lados del pomo de una puerta al eje de un motor tipo TT, y por el otro lado a un contenedor de llave universal, el cual contiene la llave de la puerta. El bloqueo inteligente se produce en el motor tipo TT gracias a que contiene un sistema de engranajes y embrague para

la rotación total del eje, lo que posibilita la apertura y cierre del sistema.

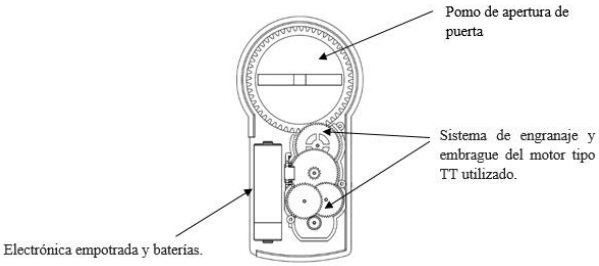


Fig. 4. Esquema del diseño del encapsulado que contiene el objeto IoT de cerradura inteligente.

El diseño final del sistema incluirá la parte electrónica y baterías, en el interior del encapsulado de la cerradura inteligente. Este encapsulado se fabricará con una impresora 3D para la demostración.

D. Integración del hardware

Para el diseño del sistema se ha optado por realizar una integración de varias placas que contienen la electrónica necesaria para controlar el motor de la cerradura inteligente. Los materiales empleados fueron: (i) Wemos D1 mini Pro, (ii) Motor de reducción tipo TT 5V, (iii) Driver Motor, (iv) Batería Li-Po 12V para alimentación del motor, (v) Cargador de batería 12V y (vi) Regulador 12V-3.3V empleado para la alimentación del microcontrolador ESP8266 de la placa Wemos D1 mini Pro.

A continuación, se describen las características de las dos placas utilizadas: Placa Wemos D1 mini Pro y Placa de drivers.

- Placa Wemos D1: Esta placa de bajo costo contiene un microcontrolador ESP8266 de 32 bits a 80 MHz (puede subirse a 160 MHz) con 64 kb+96 kb de RAM y 4 Mb de Flash con un microchip Wi-Fi de bajo costo que incorpora una pila TCP/IP completa. Para este caso en particular se ha utilizado la última versión mini Pro que contiene múltiples pines GPIO (General Purpose Input Output) para conectar diversos periféricos a la placa y que además son capaces de generar comunicaciones serie PWM (Pulse-Width Modulation), I2C (Inter-Integrated Circuit), SPI (Serial Peripheral Interface) y UART (Universal Asynchronous Receiver-Transmitter).
- Placa driver motores. La placa contiene un controlador de motores L298N que tiene dos puentes H con lo que es posible controlar la velocidad y dirección de dos motores de corriente continua o un motor paso a paso.

Para implementar el diseño preliminar del sistema, se implementó un prototipo para probar y validar la funcionalidad del sistema. Este prototipo se rige por las conexiones descritas en el diagrama mostrado en Fig. 5. Asimismo, la implementación del circuito diseñado a partir de estas conexiones dio como resultado el prototipo de hardware mostrado en Fig. 6.

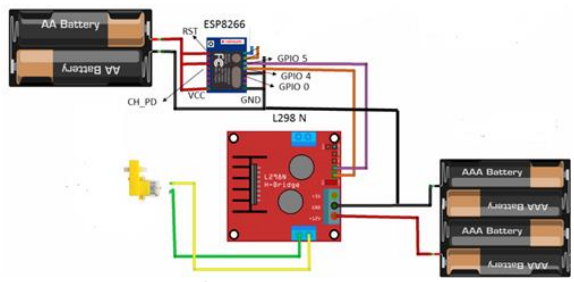


Fig. 5. Esquema electrónico.

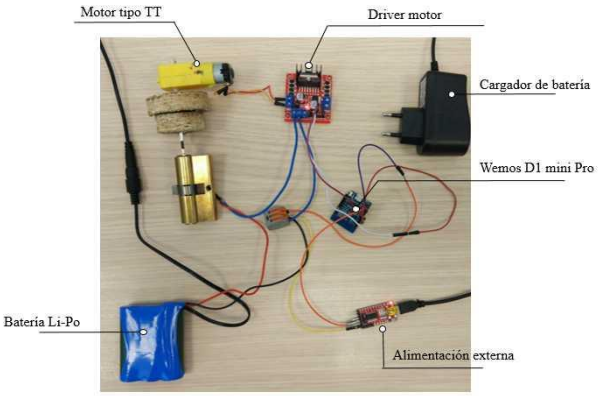


Fig. 6. Esquema electrónico.

E. Programación del firmware

El ESP8266 es un microcontrolador de código abierto en el que es posible la inyección de un firmware propio para administrar un objeto conectado como el que se describe en este trabajo. Por ese motivo, surgieron una serie de *firmware* de terceros para dispositivos IoT de control sencillo (ON/OFF, PWM), incluso con soporte web que simplifica la interacción entre el usuario y el microcontrolador del objeto IoT.

Entre los diferentes *firmware* de terceros se pueden citar los *firmware* ESPeasy, Tasmota y ESPurna [17]. Las versiones actuales de ESPeasy y ESPurna no tienen soporte de seguridad TLS. Por esto, en esta propuesta se usa Tasmota (versión 6.5.0), que soporta cifrado TLS y permite establecer conexiones MQTT seguras entre clientes y un bróker Mosquitto.

Una vez descargado el software de Tasmota para la ESP8266, se procede a modificar el firmware para incluir la configuración del objeto IoT de manera que pueda funcionar con la pasarela residencial. Para ello, hay que tener en cuenta que el espacio disponible para el firmware es muy limitado. Por defecto, se ha desarrollado un nuevo firmware que tiene las siguientes funcionalidades adicionales: (i) activación del modo TLS, (ii) activación del modo de servidor web, (iii) configuración del *firmware*. Ésta última funcionalidad consiste en establecer el nombre del dispositivo, el SSID de la red Wi-Fi local y su contraseña, la dirección IP del bróker MQTT, el usuario y contraseña de la cuenta del bróker, el protocolo de seguridad TLS y el puerto MQTT.

Para programar el firmware del módulo se realizó la conexión a un ordenador. Desde éste se actualizó el *firmware* utilizando un adaptador FTDI (*Future Technology Devices International*) a USB (*Universal Serial Bus*).

Para la apertura y cierre de la cerradura inteligente tenemos que asociar dos salidas analógicas que

determinan el sentido de giro del motor, horario o antihorario respectivamente, en base a su activación o no. Para ello, se configurado dos patillas GPIO con PWM (GPIO4 y GPIO5) para emular salidas analógicas, lo que nos permite además controlar también la velocidad del motor. Así, un valor de 100 en GPIO4 y un valor de 0 en GPIO5 implican la apertura de la puerta en sentido horario que se abre completamente si se mantiene durante 5 segundos.

Para controlar la cerradura electrónica el objeto IoT se conecta por MQTT al bróker MQTT de la pasarela residencial. El objeto IoT actúa como publicador MQTT en el tópico *status/Homelock/POWER* con el cual podemos saber si la puerta está abierta o cerrada. Por otra parte, actúa como suscriptor en el tópico *cmnd/Homelock/POWER*, de tal modo que, cuando un cliente publica un valor ON en dicho tópico, el objeto IoT ejecuta el comando para abrir la puerta activando GPIO4 con una velocidad determinada y, cuando el cliente publica un valor OFF en dicho tópico, la puerta se cierra activando GPIO5 con la misma velocidad.

El firmware proporciona un servidor web como se muestra en la Fig. 7 (izquierda) con el que se puede programar las acciones a realizar sobre el objeto IoT, así como asociar los tópicos correspondientes. Por otra parte, como se muestra en la Fig. 7 (derecha) con el programa *MQTT.fx* se puede publicar valores sobre los tópicos del dispositivo para probar su funcionamiento.

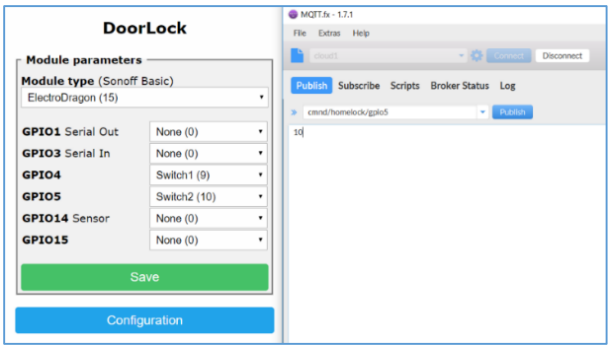


Fig. 7. Configuración de GPIO.

VII. INTEGRACIÓN DEL OBJETO EN EL ECOSISTEMA IOT

A. *La pasarela residencial*

Como pasarela residencial, además del *router* para permitir el intercambio de mensajes entre la red local y la red de banda ancha, es necesario considerar un sistema empotrado con mayores prestaciones porque se requiere instalar un bróker MQTT que facilite la transmisión entre los objetos IoT, la pasarela residencial en MQTT, y los clientes que utilizan el dispositivo móvil.

En este caso, se ha instalado el bróker Mosquitto sobre una Raspberry Pi 3 con sistema operativo Raspbian Stretch (una distribución de Debian/Linux).

Asimismo, para realizar la instalación y configuración del bróker MQTT se recomienda seguir los siguientes pasos: (i) instalación del bróker MQTT, (ii) generación de un certificado firmado con la contraseña del bróker, y (iii) obtención del *fingerprint* e integración en el firmware del microcontrolador.

Una vez instalado el *bróker* Mosquitto, se debe tener en cuenta que los ficheros de configuración incluyan los datos de usuario y contraseña del bróker, cifrados, y crear certificados TLS para asegurar la conexión MQTT a través del bróker. El fichero que contiene el usuario y contraseña está, por tanto, cifrado, y se ubica en */etc/mosquitto/user*.

El *fingerprint* emitido por la autoridad de certificación se obtiene con la ejecución de la línea de código siguiente:

```
openssl s_client -connect localhost:8883 < /dev/null 2
>/dev/null | openssl x509 -fingerprint
-noout -in /dev/stdin
```

Este *fingerprint* es una palabra de 20 bytes que se debe integrar en la fase de actualización del firmware en el microcontrolador ESP8266 para que se establezca una conexión segura con el Mosquitto.

También sería posible utilizar como pasarela residencial un servidor Mosquitto administrado en una nube externa, tal y como ocurre en muchos objetos IoT. Como ejemplo, la nube *CloudMQTT* (https://www.cloudmqtt.com/) proporciona una instancia que puede configurarse para implementar el bróker MQTT de forma remota.

B. *Programación del dispositivo móvil*

Para el acceso desde un dispositivo móvil vamos a utilizar una aplicación, *IoT MQTT Panel* disponible en el *PlayStore* de Android y en IOS, que nos permite administrar y visualizar objetos IoT a través del protocolo MQTT. La aplicación *IoT MQTT Panel* no sólo está diseñada para visualizar los diversos estados del objeto IoT, sino también nos permite organizar estas conexiones, dispositivos, mensajes, etc. La Fig. 8, representa la arquitectura básica de una aplicación típica de Smart Home.

La configuración en la herramienta *IoT MQTT Panel* del dispositivo IoT y del bróker correspondiente se hace de forma similar al firmware, pero mediante la interfaz mostrada en

Fig. 9.

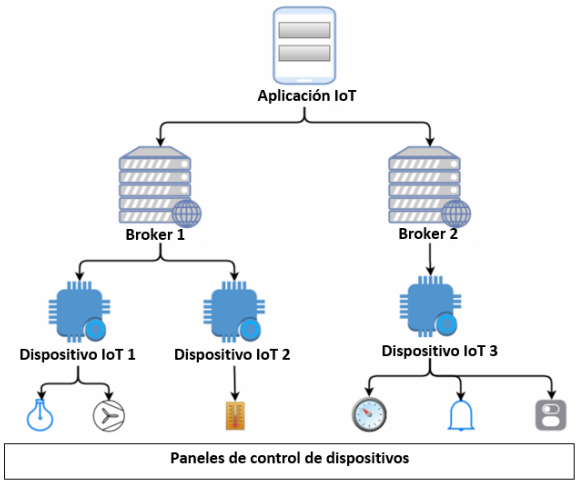


Fig. 8. Organización de los elementos de IoT MQTT Panel.

VIII. INSTALACIÓN DEL OBJETO IOT EN UN ENTORNO REAL

Una vez que se ha construido y desarrollado el nuevo objeto IoT y se ha programado su *firmware* y, por último se ha integrado en el ecosistema IoT a través de la pasarela residencial o remota, se procede a ejecutar el proceso para la instalación del nuevo objeto. Estas acciones las debe ejecutar un cliente como publicador MQTT en el tópico correspondiente.

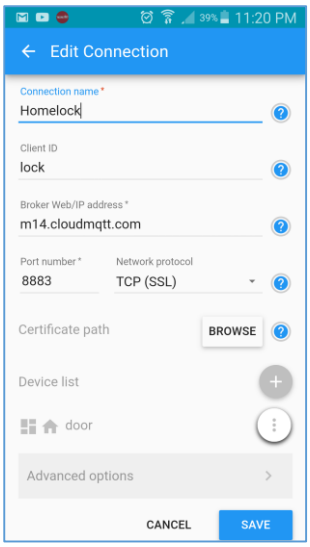


Fig. 9. Herramienta de configuración de la aplicación IoT MQTT Panel.

Como en cualquier objeto IoT comercial, el proceso de configuración del cliente que adquiere el producto debe ser lo más sencillo posible.

En una primera fase el objeto está en modo AP (Punto de acceso), lo que le permite al cliente ingresar la información del SSID y las contraseñas de los diferentes puntos de acceso a Internet. Si tiene éxito, la cerradura inteligente entrará en modo cliente. En las primeras cuatro secuencias de Fig. 10 se muestra las interacciones requeridas por el cliente.

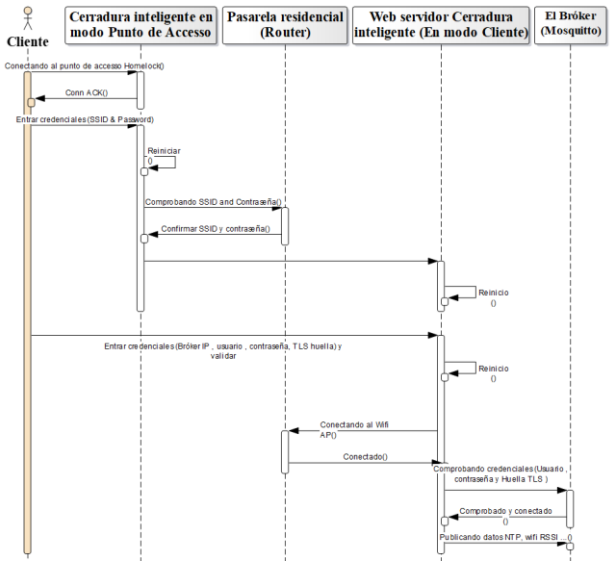


Fig. 10. Diagrama de secuencia de la configuración de la cerradura en el lado del cliente en modo cliente.

En una segunda fase, el cliente debe completar los campos relacionados con el bróker MQTT que incluye la contraseña del usuario, la dirección IP del bróker y el protocolo de seguridad TLS. En Fig. 10 se muestra un diagrama de secuencia con todas las interacciones que se producen tanto en modo AP como en modo cliente.

En la tercera fase, el cliente debe descargar y configurar la aplicación *IoT MQTT Panel*. En la Fig. 11 se muestra el conjunto de interacciones que se producen entre el cliente y el resto de elementos que conforman el ecosistema IoT. A partir de ese momento el cliente podrá disfrutar de las funcionalidades del nuevo objeto IoT de cerradura inteligente.

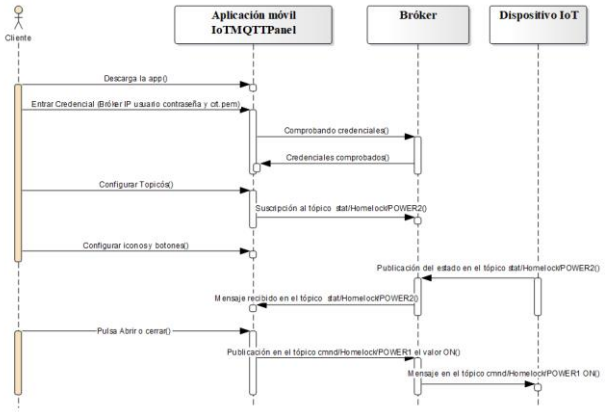


Fig. 11. Diagrama de secuencia de la configuración de la aplicación en el lado del cliente.

Una vez terminado todo el proceso, el usuario podrá abrir y cerrar la cerradura inteligente utilizando el protocolo MQTT mediante el comando ON/OFF (ON para abrir y OFF para cerrar) publicando el tópico correspondiente, esto es, *cmnd/HomeLock/POWER*. La respuesta de su estado se recibe en el tópico *stat/Homelock/POWER* (Fig. 12).

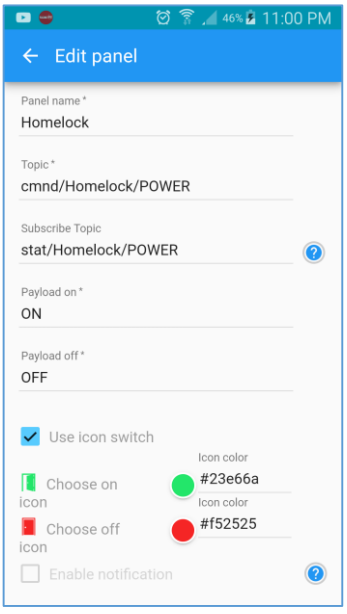


Fig. 12. Configuración de los órdenes de abrir/cerrar en el IoT MQTT Panel.

IX. CONCLUSIONES

En este artículo se ha propuesto una metodología y las etapas del proceso de diseño y desarrollo que se han seguido para la construcción de un objeto IoT y su posterior incorporación en ecosistemas de IoT de una forma segura, garantizando asimismo la escalabilidad y flexibilidad del sistema resultante para cualquier tipo de objeto IoT. La implementación del intercambio de mensajes mediante el protocolo MQTT basado en suscripción/publicación facilita un acceso rápido al estado y a los comandos del objeto IoT, permitiendo además llevar a cabo las transmisiones sobre canales seguros basados en TLS.

Como caso de estudio se ha diseñado y construido un objeto IoT, la **cerradura inteligente** siguiendo los pasos y metodologías de diseño presentadas obteniendo como resultado un objeto integrado dentro de un ecosistema IoT para Hogar Digital. Para analizar la validez de la propuesta se presentó igualmente también los detalles relativos a la configuración del objeto IoT por parte del cliente.

Dado que la seguridad es un aspecto crucial en el desarrollo de la nueva generación de dispositivos IoT, consideramos que este aspecto tiene que ser tenido en cuenta desde la fase de concepción del nuevo dispositivo IoT, y seguir recomendaciones como las que recientemente han propuesto por parte del gobierno inglés en sus código de buenas prácticas del diseño seguro para los desarrolladores de dispositivos IoT de electrónica de consumo [18].

REFERENCIAS

[1] S. Li, L. Da Xu, and S. Zhao, “The internet of things: a survey,” Inf. Syst. Front., vol. 17, no. 2, pp. 243–259, Apr. 2015.

[2] D. Singh, G. Tripathi, and A. J. Jara, “A survey of Internet-of-Things: Future vision, architecture, challenges and services,” in 2014 IEEE World Forum on Internet of Things (WF-IoT), 2014, pp. 287–292.

[3] D. Han and J. Lim, “Smart home energy management system using IEEE 802.15.4 and zigbee,” IEEE Trans. Consum. Electron., vol. 56, no. 3, pp. 1403–1410, Aug. 2010.

[4] J. Byun, B. Jeon, J. Noh, Y. Kim, and S. Park, “An intelligent self-adjusting sensor for smart home services based on ZigBee communications,” IEEE Trans. Consum. Electron., vol. 58, no. 3, pp. 794–802, Aug. 2012.

[5] D. J. Cook et al., “MavHome: an agent-based smart home,” in Proceedings of the First IEEE International Conference on Pervasive Computing and Communications, 2003. (PerCom 2003)., pp. 521–524.

[6] M. U. Farooq, M. Waseem, and A. Khairi, “A Critical Analysis on the Security Concerns of Internet of Things (IoT).”

[7] M. H. Miraz, M. Ali, P. S. Excell, and R. Picking, “A review on Internet of Things (IoT), Internet of Everything (IoE) and Internet of Nano Things (IoNT),” in 2015 Internet Technologies and Applications (ITA), 2015, pp. 219–224.

[8] V. Issarny et al., “Service-oriented middleware for the Future Internet: state of the art and research directions,” J. Internet Serv. Appl., vol. 2, no. 1, pp. 23–45, Jul. 2011.

[9] P. Pico-Valencia and J. A. Holgado-Terriza, “Agentification of the Internet of Things: A systematic literature review,” Int. J. Distrib. Sens. Networks, vol. 14, no. 10, p. 155014771880594, Oct. 2018.

[10] L. Da Xu, W. He, and S. Li, “Internet of Things in Industries: A Survey,” IEEE Trans. Ind. Informatics, vol. 10, no. 4, pp. 2233–2243, Nov. 2014.

[11] D. Chen, G. Chang, L. Jin, X. Ren, J. Li, and F. Li, “A Novel Secure Architecture for the Internet of Things,” in 2011 Fifth International Conference on Genetic and Evolutionary Computing, 2011, pp. 311–314.

[12] K. Chen et al., “Internet-of-Things Security and Vulnerabilities: Taxonomy, Challenges, and Practice,” J. Hardw. Syst. Secur., vol. 2, no. 2, pp. 97–110, Jun. 2018.

[13] A. Gai, S. Azam, B. Shanmugam, M. Jonkman, and F. De Boer, “Categorisation of security threats for smart home appliances,” in 2018 International Conference on Computer Communication and Informatics (ICCCI), 2018, pp. 1–5.

[14] J. M. Batalla, A. Vasilakos, and M. Gajewski, “Secure Smart Homes,” ACM Comput. Surv., vol. 50, no. 5, pp. 1–32, Sep. 2017.

[15] M. C. VEGA, P. O. VIVAS, C. M. RIOS, C. G. LUIS, B. C. MARTÍN, and A. H. SECO, “Las tecnologías IoT dentro de la industria conectada 4.0.” 2015.

[16] M. A. Razzaque, M. Milojevic-Jevric, A. Palade, and S. Clarke, “Middleware for Internet of Things: A Survey,” IEEE Internet Things J., vol. 3, no. 1, pp. 70–95, Feb. 2016.

[17] X.Perez,“Free Libre Open Source Firmware for ESP8266-based Smart Devices.”,Date Accesed: 11-May-2019. [Online]. Available: https://tinkerman.cat/uploads/espurna.esp.pdf

[18] J. M. Blythe, N. Sombatruang, and S. Johnson, “What security features and crime prevention advice is communicated in consumer IoT device manuals and support pages?", 08-Apr-2019. [Online]. Available: https://academic.oup.com/cybersecurity/article/5/1/tyz005/5519411”