UNIVERSIDAD DE EXTREMADURA

TESIS DOCTORAL

Optimización de Aplicaciones Científicas y de Aprendizaje Automático en Entornos de Altas Prestaciones Heterogéneos

SERGIO MORENO ÁLVAREZ

Programa de doctorado en Tecnologías Informáticas

Con la conformidad:

La conformidad del director/es de la tesis consta en el original en papel de esta Tesis Doctoral

Dr. Juan Antonio Rico Gallego *Dr. Juan Mario Haut Hurtado*

November 2021

Este trabajo no habría sido posible sin el apoyo de toda mi familia. Esta tesis es más vuestra que mía. Estos agradecimientos van por vosotros.

A mis hermanos, Mario y Daniel, por vuestra compañía, ayuda incondicional y apoyo en los momentos más difíciles.

A mis padres, Julia y Pablo. ¿Qué os voy a decir a vosotros?. Gracias por todo, vuestro trabajo y esfuerzo ha sido lo que nos ha llevado hasta aquí. Vosotros sois los verdaderos creadores de esta tesis.

Por último, a los directores y tutor de esta tesis. Juan Antonio, Mario y Juan Carlos gracias por vuestros consejos, dedicación y confianza depositada en mi.

Acknowledgements

This thesis was supported by the European Regional Development Fund 'A way to achieve Europe' (ERDF), the Extremadura Local Government and the Ministery of Economy, Science and Digital Schedule (Ref. IB20040). Project is "Machine Learning Techniques for High-Performance Computing Platforms Optimization and Remote Sensing Applications".

JUNTA DE EXTREMADURA

Consejería de Economía, Ciencia y Agenda Digital

Also, this thesis was supported by the project PID2019-110315RB-I00 / AEI / 10.13039 / 501100011033, Ministry of Science, Innovation and Universities, State Agency and FEDER Funds (European Union). Project is "Development of Deep Learning Techniques for the Optimization of Super-Computing Infrastructures and Hyper-Spectral Imaging Applications".

Unión Europea

GOBIERNO DE ESPAÑA

MINISTERIO DE CIENCIA, INNOVACIÓN Y UNIVERSIDADES

Abstract

The improvement of high-performance computing platforms has leveraged the acceleration and optimization of computationally intensive applications. Since these applications consume a large amount of resources and time, its optimization has been a key point of research. Some examples of these applications are classical scientific applications, image processing and deep learning techniques.

In high-performance computing platforms the optimization of these applications have been addressed using multiple techniques. In this sense, the workload distribution technique is widely used. This technique consists in the distribution of the workload between the processes deployed on the different computing devices that compose the platform. Current distributed applications usually perform a homogeneous workload partitioning between processes composing the application without taking into account the heterogeneous features of the resources in which they execute. As a consequence, non-optimal partitioning leads to longer execution times. These execution times are established by the slowest process. Thus, a heterogeneous distribution according to the capabilities of each process is needed. In order to achieve an optimal workload distribution it is necessary to model the heterogeneity of the platform resources. The analytical computation and communication models have traditionally been used for modeling that resources capabilities.

This thesis proposes different methodologies with the objective of improving the performance of high computational cost applications in heterogeneous platforms. The idea is to characterize the computational capabilities of the processes involved in the execution of the applications, and then perform a partition and distribution of the workload heterogeneously in terms of such capabilities. To evaluate our proposal, experiments with common scientific kernels and neural network based applications are performed to illustrate and demonstrate the advantages with respect to the current techniques available in the literature.

Resumen

El desarrollo de las plataformas de computación de alto rendimiento ha potenciado la aceleración y optimización de las aplicaciones de cálculo intensivo. Dado que estas aplicaciones consumen una gran cantidad de recursos y tiempo, su optimización ha sido objeto de investigación. Algunos ejemplos de estas aplicaciones son aplicaciones científicas clásicas, procesamiento de imágenes y técnicas de aprendizaje profundo.

En este tipo de plataformas, la optimización de estas aplicaciones se ha abordado mediante múltiples técnicas. En este sentido, el procesamiento distribuido es ampliamente utilizado para repartir la carga de trabajo entre todos los procesos desplegados en los diferentes dispositivos de cómputo que componen la plataforma. Generalmente, las aplicaciones distribuidas actuales suelen realizar un reparto homogéneo de la carga de trabajo sin tener en cuenta las características de los recursos en los que se ejecutan. Como consecuencia, una partición no óptima conduce a tiempos de ejecución que dependen únicamente del proceso más lento. Por lo tanto, es necesario adaptar la carga de trabajo en función de las capacidades de cada proceso. Para ello, es necesario modelar la heterogeneidad de los recursos de la plataforma. Los modelos analíticos de cómputo y comunicación se han utilizado tradicionalmente para establecer las capacidades de dichos recursos.

En esta tesis, se proponen diferentes metodologías con el objetivo de mejorar el rendimiento de las aplicaciones de alto coste computacional en plataformas heterogéneas. Para ello es necesario caracterizar las capacidades computacionales de los procesos que intervienen en la ejecución de las aplicaciones, para posteriormente realizar una partición y distribución óptima. Nuestra propuesta ha sido evaluada mediante experimentos con *kernels* científicos comunes y aplicaciones basadas en redes neuronales, ilustrando así los beneficios de las técnicas propuestas con respecto al estado del arte.

Contents

Compendium of publications

The following publications compose this presented thesis as a compendium:

1. **Moreno-Alvarez, S.**, Haut, J. M., Paoletti, M. E., Rico-Gallego, J. A., Diaz-Martin, J. C. and Plaza, J. (2020). Training deep neural networks: a static load balancing approach. The Journal of Supercomputing, 76(12), 9739-9754. [IF(2020)=2.474].

2. **Moreno-Alvarez, S.**, Haut, J. M., Paoletti, M. E. and Rico-Gallego, J. A. (2021). Heterogeneous model parallelism for deep neural networks. Neurocomputing, 441, 1-12. [IF(2020)=5.719].

3. Paoletti, M. E., Tao, X., Haut, J. M., **Moreno-Alvarez, S.** and Plaza, A. (2021). Deep mixed precision for hyperspectral image classification. The Journal of Supercomputing, 1-12. [IF(2020)=2.474].

4. Haut, J. M., Paoletti, M. E., **Moreno-Álvarez, S.**, Plaza, J., Rico-Gallego, J. A. and Plaza, A. (2021). Distributed Deep Learning for Remote Sensing Data Interpretation. Proceedings of the IEEE. [IF(2020)=10.961].

Chapter 1

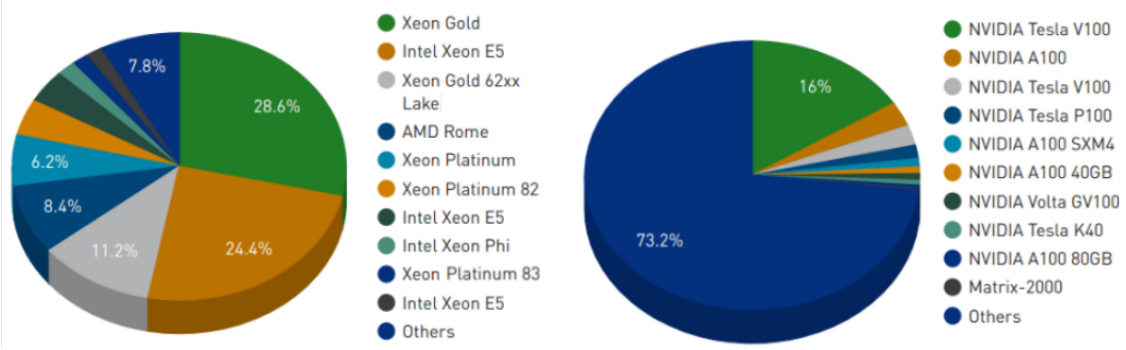
General overview

1.1. Challenges of heterogeneous supercomputers

High Performance Computing (HPC) platforms have been used to tackle with high computational demanding applications as linear algebra [78], Neural Networks (NN) [61, 28, 11], data mining [30] and Machine Learning (ML) algorithms, including classification [69, 58, 40], regression and clustering [29]. These applications are deeply studied in Chao Wang *et al.* [85]. The main reason of the HPC platforms usage is that they provide a big amount of fast processing computational resources, distributed file systems and high speed networks to communicate data and results between devices. Applications executed on HPC platforms are composed of processes deployed on the platform resources that collaborate to solve a problem. Therefore, processes use the computational and communication resources available in the platform to perform computations and exchange results.

From a different perspective, the high computational and communication needs of the applications have leveraged the improvements in the capabilities of the devices, allowing the advancement of HPC environments which are continually improving their resources. As a consequence, central processing units (CPUs), graphic processing units (GPUs), tensor processing units (TPUs) or embedded devices, along with high-speed communication networks such as Infiniband or Ethernet, are included in these platforms. In the Top-500 [27] list of supercomputers, the evolving heterogeneity is appreciated. The Top-500 is a ranking list of supercomputers all over the world which is updated twice every year. For example, the JUWELS supercomputer [35] is found on June 1st, 2021 in the position 8th of this ranking. This platform is composed of two modules, Cluster and Booster. The Cluster module provides general purpose computing with more than 2300 nodes made up of Skylake CPUs, while the Booster module is composed of by 936 nodes with 3744 NVIDIA A100 GPUs. Both modules can be used together, providing a large amount of heterogeneous

resources. Indeed, system provides with different available networks connections between nodes, including the 200Gbit/s Mellanox HDR200 InfiniBand network.



a) CPUs System Share b) GPUs System Share

Figure 1.1 Detailed percentage values for Top500 heterogeneity statistics (June 2021).

In the Top500 ranking, the heterogeneity of the platforms is observable. In the recent years, heterogeneous systems composed of nodes with CPUs+GPUs has dominated the TOP500 ranking due to better performance. There are some exceptions of homogeneous platforms composed of CPUs as Fugaku, Sunway TaihuLight, K-computer or Frontera.

The resulting heterogeneity of supercomputing platforms is a challenge to the achievement of good performance of applications. Regarding the capabilities of the resources, there are multiple works which have studied particular aspects of these components such as memory bandwidth [79], power consumption [20, 68], on-board processing [52], performance [81] or resource management [2, 82]. Figure 1.1 shows the heterogeneity of CPUs and GPUs types and its usage rate in the Top500 ranking. Together with the continuous increase in the number of devices in the platforms, also known as scaling, previous factors play a crucial role in the application efficiency.

Scaling promotes the use of more devices to execute an application, which means a heterogeneity increment. The scaling trends and issues for supercomputing platforms are studied in [93, 26]. A scaling example analyzing the application speed and providing a better understanding of the computation time is provided in [73]. It describes scalability for CPUs and GPUs platforms using different communication links. Meanwhile, the work [54] studies the future impact of heterogeneous scaling for diverse applications and approaches. The advances during the recent years in these computing devices architectures (CPUs and GPUs) are shown in the Table 1.1. It is clearly observable the development trend of architectures in short time periods. In this aspect, there exist multiple works and studies that evaluate the performance of different GPU and CPU architectures [15, 19, 74], demonstrating the acceleration and performance of every architecture generation.

Year	Type	Device	Architecture	GFLOPS	Cores	Memory	Frequencie
2011	CPU	Xeon 5690	Westmere	166	6	32	3470
2011	GPU	Tesla M2090	Fermi	1331	16	177	1300
2012	CPU	Xeon E5-2690	S.Bridge	372	8	51	2900
2014	CPU	Xeon E5-26993	Haswell	1324	18	68	2300
2015	GPU	Tesla K40	Kepler	5040	15	288	745
2015	GPU	Tesla M40	Maxwell	6844	24	336	1000
2016	GPU	Tesla P100	Pascal	9340	56	720	1328
2017	CPU	Xeon Plat8180	Skylake	4480	28	120	2500
2017	GPU	Tesla V100	Volta	14899	80	900	1380
2018	CPU	Xeon Plat9282	C. Lake	9320	56	175	2600
2020	GPU	Ampere A100	Ampere	19500	108	1555	1410

Table 1.1 Different compute architectures over the years for CPU and GPU devices. Memory is shown in bandwidth (BW) units and frequency in (Mhz). Cores per GPU are streaming multiprocessors (SM).

HPC allows the execution of parallel applications, providing the usage of a big amount of processes deployed on the platform resources. Processes are in charge of executing data parallel applications in a sequence of computation and communications steps. Thus, one of the main challenges in current heterogeneous HPC platforms is to find an optimal workload balance between processes. The simplest approach is to distribute the computational workload in a equitable mode (homogeneously), that is, not considering the resources features of the processes. Hence, processes are assigned with the same workload amount. As a consequence, in the communication step, faster processes that have completed the computation step should wait at synchronization points for the rest of the processes, which highly degrades the overall performance. Consequently, the application time is determined by the slower process. Meanwhile, faster processes finish its assigned computation earlier. The conclusion is that the distribution of the workload should consider the uneven resources computational capabilities. Heterogeneous workload distribution is based on the estimation of the processes computational capabilities, assigning to each process a workload amount proportional to its speed, that results in an improvement the application overall performance. Figure 1.2 shows an example of a heterogeneous platform with deployed processes assigned with different resources.

Nevertheless, heterogeneous workload distribution have some factors that should be considered. Since different amounts of workload are assigned to each process, memory capacity of each resource and process is an important factor that limits the assigned workload size to avoid memory overloads. In addition, as the workload is partitioned and processes have different computational capabilities, communication imbalances appears, that impact

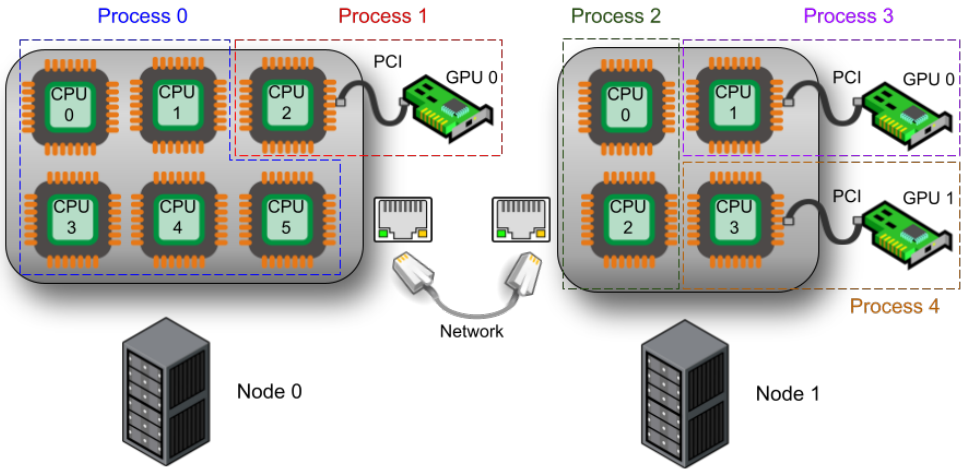


Figure 1.2 Example of a heterogeneous HPC platform composed of 2 Nodes. Nodes are composed of a different number of CPU cores and GPUs. Five processes *(0-4)* are deployed in the platform and assigned with different resources.

the global execution time. Finally, data movements between processes progress through different communication channels, which are determined by the mapping of the processes on the platform resources and the application communication pattern.

Depending on the application, the communication and computation requirements may vary. As a consequence, applications running on modern HPC platforms with tightly coupled computing devices should consider the following aspects for the optimization of its running applications:

- As the performance highly degrades with the heterogeneity of the platform, applications should balance the workload considering the computational capabilities of each process, which should be determined according to its speed.
- Applications should consider memory restrictions and its representative properties to avoid unnecessary memory overloads.
- Applications should reduce communication imbalances, selecting the appropriate data flows between processes by using adequate communication channels and communication patterns.

Hence, heterogeneous workload distribution is an optimization problem that focuses on the previous three challenges, and it is the main motivation of this thesis. To obtain initial insights, the following Section 1.2 describes different scientific applications addressed in this work, also known as kernels. Then, challenges of reducing the execution time of HPC applications using communication and computation balancing are deeply studied.

1.2. Heterogeneous data partitioning for scientific kernels.

Data partitioning is a technique to distribute data workload between the processes running on a given platform computational resources. This technique is based on partitioning the data into smaller chunks or partitions. Each partition is assigned to an specific process. In high computational demanding applications, the data partitioning and distribution is critical for the performance of the applications. Traditionally, data partitioning techniques have been designed to capture the application behaviour and the heterogeneity of modern heterogeneous platforms. The objective is to determine the processes capabilities in order to find an optimal distribution of the application workload based on these capabilities.

In the literature, heterogeneous data partitioning methods have demonstrated improvements in the performance. As an example, work [36] proposes an efficient recursive sequential algorithm to explore the available solutions and to determine the parallel heterogeneous data distribution. However, most of the state-of-art works use modeling techniques to capture a simplified description of the computation and communication costs. This thesis applies this heterogeneous data partitioning method to Data-Parallel kernels.

In a data-Parallel kernels, all the processes execute the same algorithm on an assigned partition of the data space. These kernels are the main components of scientific applications running in HPC platforms. Attending to heterogeneous kernel workload partitioning, the procedure is as follows:

1. The first step is to determine the computational capabilities of the processes. Mainly, there are two techniques to determine the processes speeds. The first technique is using a numerical value representing how faster is a specific process with respect to the rest, and is called Constant Performance Models (CPM). The second technique are Functional Performance Models (FPM), which represent the speed of every process as a function dependant on the size of the problem. As a result of these techniques, the speed of the processes is obtained. FuPerMod is a tool proposed in [46], that executes a benchmark to determine the processes speeds. The benchmark is provided by the user and should be representative of the application computations. A benchmark example that could be used by FuPerMod are General Matrix Multiplications (GEMMs) to provide a good representation of the computation performed in matrix multiplications operations. In FuPerMod, the functional performance models of the processes are generated as a speed vector $S = \{s_1(x), s_2(x), \ldots, s_P(x)\}$, with P the number of processes, and x the problem size.

2. The second step is to perform the data partitions to be assigned to the processes using the speeds obtained in the previous step. A way to do this is to assign heterogeneous

partitions with a specific shape. Each process performs the computation of its assigned data. The shapes of the partitions partially determine the communications of processes.

3. The final step is to define a metric to measure the communications costs between processes. In this sense, analytical communication modeling techniques are used as a mathematical representation of the application behaviour to predict communication costs. In the analytical communication models, the most representative features of the communications are considered, including patterns specific to each kernel.

Following, an introduction of two representative HPC scientific kernels is performed. Later, the kernels communications are deeply analyzed. Chosen kernels are called SUMMA and Wave2D.

1.2.1. SUMMA kernel computation partitioning

The Scalable Universal Matrix Multiplication Algorithm (SUMMA) is a dense matrix multiplication [83] over a 2-dimension data space. The algorithm executes the multiplication $C = A \times B$, being C, A, B square matrices identically partitioned into P rectangles arranged in columns, being P the number of processes. In each iteration $k \in N$, each process update the values of its assigned blocks $c_{ij} = c_{ij} + a_{ik} \times b_{kj}$, where a_{ik} is the k^{th} column and b_{kj} is the k^{th} row, both of them with its respective blocks. Blocks are the unit used to represent computation and communication. The k^{th} row and column are named as pivot block row (*pbr*) and pivot block column (*pbc*), respectively. These pivots run through matrices A and B, from left to right (*pbc*) and from top to bottom (*pbr*). In this algorithm, communication modeling between processes is calculated with the pivots movements as shown in Figure 1.3(a). To do this, the *pbc* in matrix A is communicated from the owner processes to the horizontally overlapping processes. The process owning the *pbr* broadcasts it to the rest of processes in the column. Once the N iterations are over, each block of the resulting matrix C contains the final value $c_{ij} = \sum_{k=0}^{N-1} a_{ik} \times b_{kj}$.

1.2.2. Wave2D kernel computation partitioning

The Wave2D kernel implements a bi-dimensional wave propagation equation, that is a type of equations solved under the technique of finite differences. The goal is to mathematically model the vibrations of a membrane u over a surface z for given positions (x, y) at time t. It is formulated as $u_t = c^2 \nabla^2 u$, where $\nabla^2 = \frac{\delta^2}{\delta x^2} + \frac{\delta^2}{\delta y^2}$ is the Laplacian variable to measure the density of a continuous probability and c represent the units of velocity. Hence, the solution in the surface is calculated at a specific step $u(x, y, t)$, being $0 \leq x < a$, $0 \leq y < b$ and

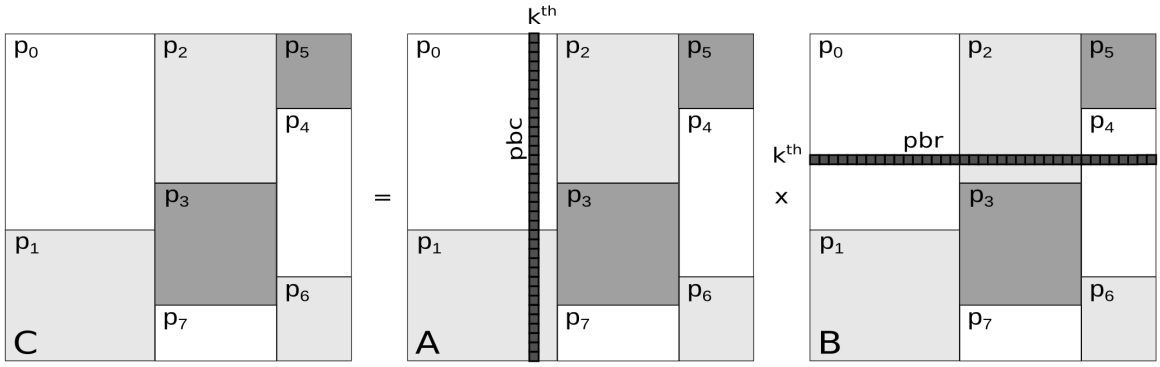


Figure 1.3 SUMMA algorithm on a heterogeneous platform with $P = 8$ processes. The figure shows the iteration k for the *pbc* and *pbr* calculating the respective value in matrix C.

$t \geq 0$ with a, b the boundaries of axis x, y, respectively. In this way, $u(x,y,t+1)$ is generated from the previous steps $u(x,y,t)$ and $u(x,y,t-1)$ to model the movement of the membrane as a wave. This behaviour is shown in the Figure 1.4 left part. A simple partition of the data is shown on the right part where the communication of the halo elements between processes p and $p+1$ is done vertically in the represented matrix. In addition, the image on the left in the Figure 1.4 represents the dot matrix that defines the wave propagation in the data space.

1.3. Analyzing kernel communications influence.

Computational partitioning approaches only consider the computation capabilities of the resources, disregarding the communication costs. Beaumont *et al.* [7, 6] sets that “The problem of partitioning dense matrices into sets of sub-matrices has received increasing attention recently and it is crucial when considering dense linear algebra and kernels with similar communication patterns on heterogeneous platforms". The communication optimization problem is NP-Complete. This means that with a high number of processes is impossible to solve in an affordable time.

The communication have a determinant influence in the overall execution time of a kernel or application. Nevertheless, workload partitioning techniques usually does not consider communications between processes, which impacts the execution time. As a consequence, even with an optimal computational load balance, the application overall performance is not optimal. Since each application have different communication pattern, as shown in Figures 1.3 and 1.4 for the SUMMA and Wave2D kernels, it is necessary to model the communication costs individually for each kernel. Minimizing the total volume of the kernel communications is not enough in these platforms due the usage of communication channels

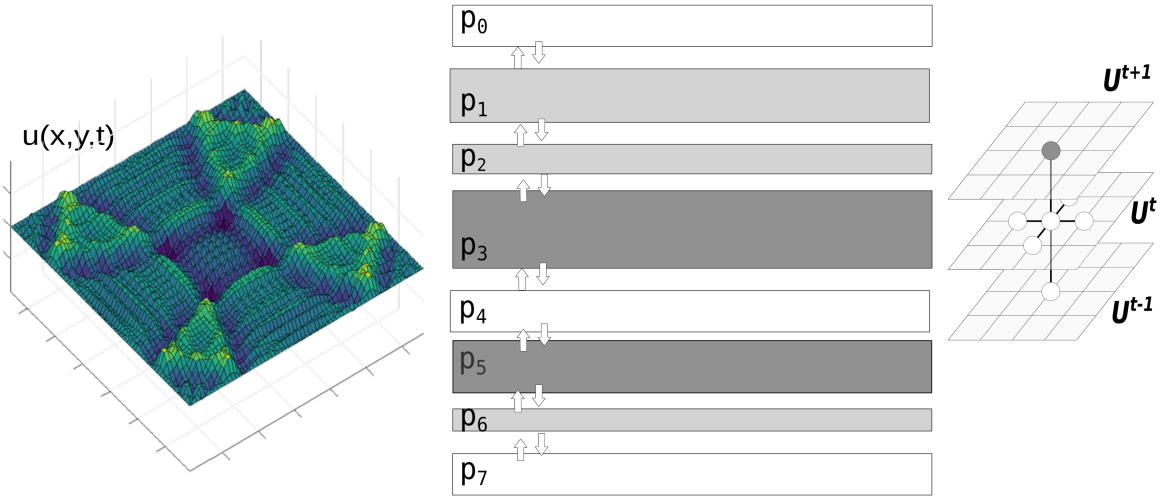


Figure 1.4 Wave2D visualization on the left of the discrete solution $u(x,y,t)$. The computation on the right side represent a 1D partition where each process $p_i \in P$ computes its slice using the halo elements from the neighbor processes $p-1$ and $p+1$. Each point of the Wave2D propagation is calculated as shown in the right side of the image.

with uneven performance. A solution is to analytically represent the cost of communication and model the optimization problem with communication cost as the objective to minimize.

Analytical communication models allow to express the application behaviour in a platform as an equation based on a set of parameters. The complexity of the analytical model increases dramatically with the heterogeneity of the platform. Multiple works have been proposed with the objective of describe the communications accurately to simplify the communication optimization problem. Approaches LogGP [9] and its heterogeneous counterpart HLogGP [10] proposed point-to-point (P2P) models to describe the communications. The LMO model [45] is based on the Hockney model [33]. HLogGP and LMO models have limited representations of the communication since factors as the topology and contention are ignored. Contention represents the number of concurrent transmission in the same communication channel. The model $\tau - Lop$ [67, 66, 65] have been proposed in the recent years. This model aims to implement an automatic model generation if communications with simple cost expressions and contention awareness. Additionally, the $\tau - Lop$ model captures the concurrency of sequence transmissions from every process while LMO and HLogGP do no take into account this factor. Also, this tool defines every sequence transmission as a unique transmission between the source and destination process with the size of the message m_j through the channel c_j. Hence, each sequential transmission is modeled as $1 \parallel T^{c_j}(m) = [o^c(m) + L^{c_j}(m,1)]$, being the overhead o the time elapsed since the message is built and sent and L the transfer time of a message through the channel c_j. The operator

$\|$ represents the cost of concurrent transmissions. Concurrent transmission are modeled as $\tau \| T^{c_j}(m) = [o^c(m) + L^{c_j}(m,\tau)]$ with τ concurrent messages transmissions through the channel c_j.

1.3.1. Kernels communication modeling using the $\tau - Lop$ model

This section models the communications for the previously introduced SUMMA and Wave2D kernels using the $\tau - Lop$ analytical communication model.

The SUMMA communication modeling is realized using the $\tau - Lop$ approach between processes. The sending and receiving blocks of the *pbc* and *pbr* can be performed using different communication patterns, such as P2P transmissions involving two processes, ring communication between processes in the same row or column, and broadcast pattern involving a set of processes in the same column. As example, modeling the P2P pattern of communication is outlined below for the SUMMA Kernel using $\tau - Lop$ (Θ), where Θ represents the communication cost. First, the horizontal cost of sending pbc blocks is represented in matrix A. The cost of the sequence of transmissions performed by each process p holding the *pbc* to its overlapping process is represented as $\Theta^{pbc}_{k,p}$ at iteration k. The network channel c_j perform the transmissions of the messages of size m_j. Hence, being P^{pbc}_k the set of processes holding the *pbc*, the global communication cost is $\Theta^{pbc}_k = \|_{p \in P^{pbc}_k} \Theta^{pbc}_{k,p}$, with $\|$ representing the concurrent transmissions. Similar, the description of the vertical communications of sending the *pbr* blocks in matrix B, where P^{pbr}_k is the set of all processes holding the *pbr* in iteration k defines the global communication as $\Theta^{pbr}_k = \|_{p \in P^{pbr}_k} \Theta^{pbc}_{k,col(p)}$ to the rest of the processes in the same column. Hence, the Equation 1.1 defines the total communication cost of the SUMMA approach:

$$\Theta^{SUMMA} = \sum_{k=0}^{N-1} [\Theta^{pbc}_k + \Theta^{pbr}_k] \tag{1.1}$$

Regarding the Wave2D kernel, it communicates intermediate results using P2P transmissions. As an example, P2P transmission T from process p_1 to its neighbours are shown in Figure 1.3(b). Assuming a computational balance between the data assigned to the processes, all processes start their corresponding transmission at once after the computation phase. Thus, we can define the cost per iteration for a specific process p with η_p neighbour processes as $\Theta_p = \|_{j \in \eta_p} T^{c(j)}(m(j))$, being $T^{c(j)}$ the P2P transmission cost through channel $c(j)$. As all P processes executes in parallel and hence, the communication is also parallelized, the

total communication of the Wave2D kernel for K iterations is determined as formulated in Equation 1.2.

$$\Theta^{W2D} = K \times [\mathop{\|}_{p=1}^{P} \Theta_p] \tag{1.2}$$

1.4. Communication optimization in data-parallel applications.

Data-parallel applications executes the same algorithm in all processes with disjoint data partitions. Processes communicate with others intermediate results. Thus, communication metrics are implemented to reduce the communication cost in data-parallel applications.

The aforementioned advances in partitioning the workload between processes have led to the need of optimizing communications between processes. Different strategies based on partitioning the workload in heterogeneous platforms have been proposed in the literature. As a example, Beaumont *et al.* [7] proposed a load-balanced algorithm which creates a column-based partition to minimize the volume of communications. A communication metric is applied based on the perimeter of the rectangles, and hence, it is minimized when rectangles become as square as possible. Following, the original formulation of the problem is exposed to later apply the $\tau - Lop$ tool as a optimization metric.

The algorithm assumes as inputs, P processes, a 2D data space of size $N \times N$ and the computational definition of the processes as a relative speed vector $S = \{s_1, s_2, \ldots, s_p\}$. Each process will be assigned with its own non-overlapping tile of the data space with size $n_p = w_p \times h_p, 1 \leq p \leq P$ with $\sum_{p=1}^{P} n_p = 1$, being n_p the data size proportional to the speed $\frac{n_1}{s_1} = \frac{n_2}{s_2} = \cdots = \frac{n_p}{s_p}$ of process p. The algorithm minimizes the half-perimeter objective function $\beta = \sum_{p=1}^{P}(w_p + h_p)$, and hence, minimize the overall communication.

The proposal generates an arrangement adding processors as row or column following three steps. First, the relative speed S vector is ordered upwardly, being the slowest process located in the first position of the vector $s_1 \leq s_2 \leq \cdots \leq s_p$. Secondly, each step of the algorithm adds a new p rectangle following the order described in S. Such new rectangle could be added to the last column (and hence resizing that column) or as a new column. Finally, possible arrangements of the new rectangle are evaluated to find the optimal distribution using the β metric. An example of the calculation process for the β metric is shown in Figure 1.5. To have a better understanding of the algorithm, the example from the Figure 1.5 is following detailed. First, the process p_1 is added as a column. Secondly, the addition of p_2 have two possibilities, as a row or as a column. In the case of adding p_2 as a column, the previous p_1 rectangle is not modified. In the case of adding p_2 as a row, the resulting

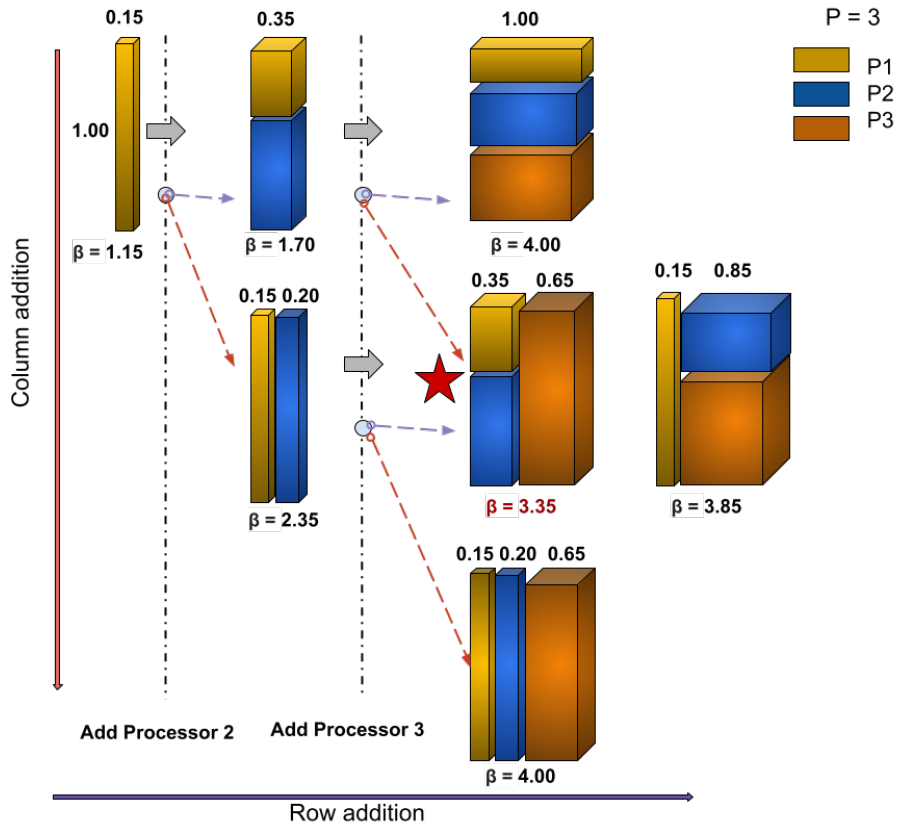


Figure 1.5 Beaumont partitioning algorithm for $P = 3$ processes with a relative speed vector of $S = 0.15, 0.2, 0.65$. Each step add a process in a column and row arrangement. The metric β outputs the value for each tiling, being the selected one the minimum value highlighted in red $\beta = 3.35$. The showing step (red star) adds the process p_3 as a column.

partition width is the sum of both columns values and the height is modified to maintain the same data amount for both processes. Finally, p_3 is added for all possibilities, as a new row or as a new column. Finally, when all processes are arranged, the β metric is calculated to select the best arrangement.

1.4.1. Communication time as optimization metric for kernels

Beaumont metric β aims to capture the communication cost of the algorithm. It generates an output partition dependant on the half perimeters communication cost. Even though the approach is able to produce a near optimal solution, it does not consider the heterogeneity of the communication networks, in which every communication progresses through a communication channel with different performance. Therefore, the following issues are determined in the application of β for the SUMMA and Wave2D kernel execution:

- Communication pattern used by the kernels are not captured. Therefore, since the communication pattern is quite different for the SUMMA and Wave2D kernels, the generated partitions should differ from each other. SUMMA communicates with processes in the same row. Meanwhile, Wave2D takes into account the nearest neighbours for each step calculation. Instead, the metric β outputs the same partitioning solution for both kernels, which is completely incorrect.

- Communication channels differences are not considered. In current heterogeneous platforms, the communication costs of different channels communicating pairs of processes differs notably in performance [16, 91]. Indeed, kernels use advanced communication facilities, including collective operations as Broadcast, Allgather and Allreduce communications [24]. The β metric only takes the communication communication volume and P2P message passing.

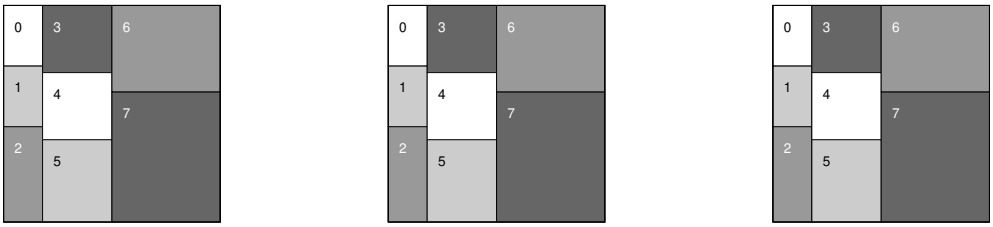
	$\Theta^{IB,P2P}_{SUMMA}$	$\Theta^{IB,Ring}_{SUMMA}$	$\Theta^{TCP,P2P}_{SUMMA}$	$\Theta^{TCP,Ring}_{SUMMA}$	$\Theta^{IB,P2P}_{W2D}$	$\Theta^{TCP,P2P}_{W2D}$	$\beta(X)$
c=1	1.73	1.80	2.33	2.69	$\mathbf{3.21^{-5}}$	$\mathbf{3.66^{-4}}$	9.00
c=2	0.64	0.64	0.83	**0.80**	3.62^{-5}	3.98^{-4}	5.76
c=3	**0.54**	**0.51**	**0.82**	0.80	4.10^{-5}	4.08^{-4}	**5.50**
c=4	0.65	0.69	0.89	1.07	4.43^{-5}	4.29^{-4}	5.88
c=5	0.87	0.97	1.15	1.42	6.15^{-5}	3.95^{-4}	6.50
c=6	1.12	1.26	1.49	1.82	3.93^{-5}	4.97^{-4}	7.28
c=7	1.14	1.57	1.88	2.25	3.60^{-5}	6.49^{-4}	8.10
c=8	1.73	1.80	2.33	2.70	$\mathbf{3.21^{-5}}$	$\mathbf{3.66^{-4}}$	9.00

Table 1.2 Cost values obtained for $\beta(X)$ metric and Θ in mili-seconds for the SUMMA and Wave2D kernels. The Θ metric is only applied in the last step with $P = 8$ processes.

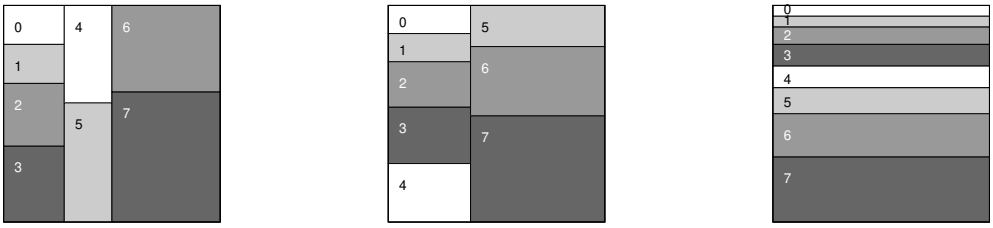
Attending to these issues, in heterogeneous platforms is critical to represent the communication accurately. The $\tau - Lop$ model is used to address this challenge with the objective of minimizing the communication cost $\Theta(k, \pi, X)$, where k is the kernel, π is the communication network and X is the available data.

Following, a comparative study of the Beaumont $\beta(X)$ and $\Theta(k, \pi, X)$ approaches is presented. The components of the study are the speed vector S = [0.05, 0.05, 0.08, 0.1, 0.1, 0.12, 0.2, 0.3] with $P = 8$ processes and $M = 4$ nodes. The networks used are Transmission Control Protocol (TCP) and Infiniband (IB). Results are shown in the Table 1.2 for the two approaches (Θ and β) with the winner partition in bold. Note that the winner partition contains $c = 3$ columns.

Obtained mapping of the β approach from Table 1.2 is Z = {0, 1, 2, 3, 0, 1, 2, 3} as shown in Figure 1.6(a), being each element of the vector, the node $Z[p] \in M$ assigned to a specific

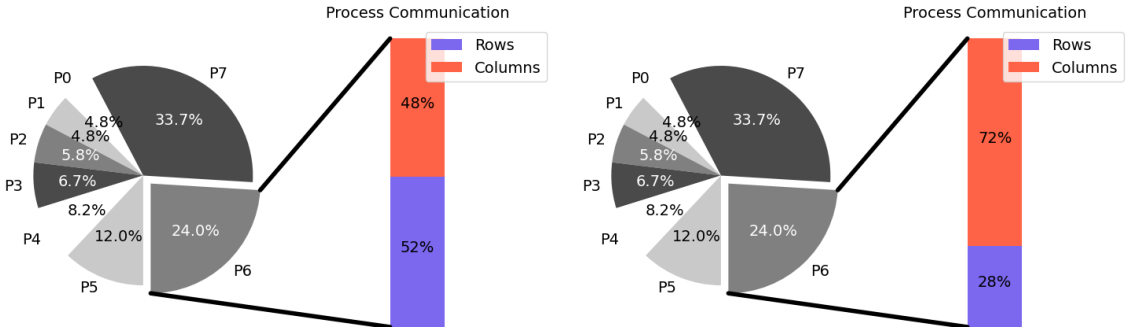


a) β partition b) $\Theta^{IB,P2P}_{SUMMA}$ c) $\Theta^{IB,Ring}_{SUMMA}$



d) $\Theta^{TCP,P2P}_{SUMMA}$ e) $\Theta^{TCP,Ring}_{SUMMA}$ f) $\Theta^{IB/TCP,P2P}_{W2D}$

Figure 1.6 Obtained optimal partitions for both metrics β and Θ metrics. Each rectangle of the mapping is assigned to a process (number).



a) β metric b) $\Theta^{TCP,Ring}_{SUMMA}$

Figure 1.7 Communication for the β and Θ metrics for the SUMMA approach. Pie chart shows the amount of data assigned to each process (%). Colors of the pie chart shows the assigned node based on Z. Bar plots shows the communication of the *pbc* and *pbr* (%). Colors from bar plots represent Figure 1.5 row and column additions.

process p. This resulting mapping is improved using the $\tau - Lop$ metric. In contrast with the β communication model which is generic to all platforms, $\tau - Lop$ should be executed in each platform to obtain the different values of the parameters o overhead and L transfer time. Based on the parameter measurement and the analytical communication modeling, $\tau - Lop$ obtains the estimation of the communication time. Obtained partitions using the $\tau - Lop$ metric are shown in Figure 1.6 for the different network channels IB and TCP. Also, different communication patterns [53] as ring-allreduce and P2P are used. The Figure 1.7 represents

the communications through rows and columns in the SUMMA kernel for β and $\tau - Lop$ approaches.

As conclusion, metrics as the proposed by Beaumont, reduce the complexity of the communications representations and obtain a good solution to reduce them. On the other hand, analytical communication models aims to capture more accurately the communication between processes for data-parallel applications in an affordable time.

1.5. Deep learning in heterogeneous HPC platforms

Deep learning (DL) have gained increasing interest in the research community due to the great advances in performance during the last decades. Focusing on this thesis, this fact has motivated the development of several optimization methods for deep learning based algorithms. In this sense, deep learning algorithms can be applied to multiple fields as image processing [58, 59], natural language processing [89] or medical approaches [43]. As a result, HPC environments have been exploited to parallelize/distribute deep learning algorithms with the aim of improving the computational performance. Table 1.3 shows the high runtime consumption conducted by the training step of deep learning models and the growing of GPUs computing speed, as shown in the previous Table 1.1.

Proposal	**Platform**	**Memory**	**Data**	**Running Time**
[41]	2 GTX 580	$2 \times$ 3Gb	1.2M HD images	6 days
[47]	1K CPUs	N/A	10M images	3 days
[18]	64 GTX 680	$64 \times$ 4Gb	10M images	3 days
[62]	GTX 280	1Gb	1M images	1 day
[22]	1K CPUs	+6Gb	1.1B audios	16 hours
[70]	96 V100	$96 \times$ 16Gb	500K MS images	0.5 hours

Table 1.3 Summary of large-scale deep learning approaches from literature. Thousands, Millions and Billions units are represented as K, M and B, respectively. MS are multi-spectral images with more than 3 channels. HD are high-definition images.

HPC offers the possibility of distribute the execution of multiple applications. Some examples of these applications are neural networks [44, 80], data mining [38], genome sequencing [51] and Deep Neural Networks (DNNs) based on deep learning architectures [63]. These applications are deeply studied in Chao Wang *et al.* [85].

Deep learning algorithms are based on the structure of a brain, which comprises a net of computational nodes called neurons. Each neuron receives multiples inputs from the initial

data and hence produce an output using the weighted sum of the inputs passed through an activation function. This function makes the learning a simpler process. Thus, to correctly learn, deep learning architectures use multiple and hierarchically-connected layers, where neurons are organized to progressively extract deep and abstract features from the data. The first layer of neurons is called *input layer*, whilst the last layer of neurons is the *output layer* and therefore the intermediate layers are named as *hidden layers*. Neurons are trained to adjust the optimal weights that minimize the loss signal produced between the desired output and the obtained one. This process is named as *forward propagation*, where at each layer j, neurons do the computation $y^j = f(\sum_{i=1}^{n} W_{ij} \times z_i + b)$, where W are the weights of the model that are adjusted to the training data to obtain a better representation of the data features, z are the activations, b is the bias and $f(\cdot)$ generates an output activation when a threshold define by the shape of the function is reached. After that, the objective is to minimize the training loss over all of the training examples to descent over the slope of the error $E = \frac{1}{2}\sum_i (\hat{y}^i - y^i)^2$, where, $\hat{y}^i$ is the true answer for the i-th training example and y^i is the value computed by the neural network. This method is known as the *gradient descent (GD)* optimization. Finally, weights parameters θ are updated following Equation 1.3.

$$\theta^{t+1} = \theta^t - \alpha \frac{\partial E(X, \theta^t)}{\partial \theta} \tag{1.3}$$

where, θ defines the set of trainable parameters of the model, X is the input data passed through the network, t is the iteration of the training procedure and α is the learning rate that modulates the behaviour of the gradient descent algorithm.

The input data of deep networks can be represented by multiples data types such as raw, images or graphs. The high potential and ability of these networks for features recognition, different ability learning (such as driving a car), natural language processing or biomedical applications has been widely proved in the literature. For instance, Dieleman *et al.* [25] have classified different galaxies types, Zhou and Troyanskaya [94] are able to predict and construct protein structures, whilst Serizel and Giulani [72] have largely studied the application of deep models for speech recognition in children and adults language. Focusing on image processing, Convolutional Neural Networks (CNNs) have been proposed to solve the spatial feature extraction from images [49, 50, 3, 56]. In this context, convolutional layers are composed of neurons, which are activated depending on certain visual stimuli, i.e. features. Thus, each layer works as a visual filter which respond to a specific and abstract stimuli. Layers can process N-dimensional arrays by locally applying multidimensional kernels over the input data, overlapping the kernel onto defined regions. Indeed, kernels K^l are filters defined as adjustable arrays with a size of $k^l \times k^l \times f^l$, which operates over the input data by performing an affine transformation between the learnable weights within K^l

and the input layer data. This transformation is indeed the element-wise product between the current features from the data window and the kernels weights, where k^l refers the spatial dimension (width and height) and f^l the spectral dimension (depth) of a specific layer l. The output of the kernel operations is a set of features maps, which are extracted by the kernels comprised by the l-th layer and which are sent to the next layer $l+1$. The workaround for forward and backward propagation is the same and it is also applied to the kernels.

1.5.1. Analyzing distributed deep learning communications

Distributed deep learning algorithm splits the input data into multiple partitions that are assigned to the processes in a distributed environment as HPC platforms. The scaling in distributed deep learning algorithms from the training speed perspectives is analyzed in the work [80].

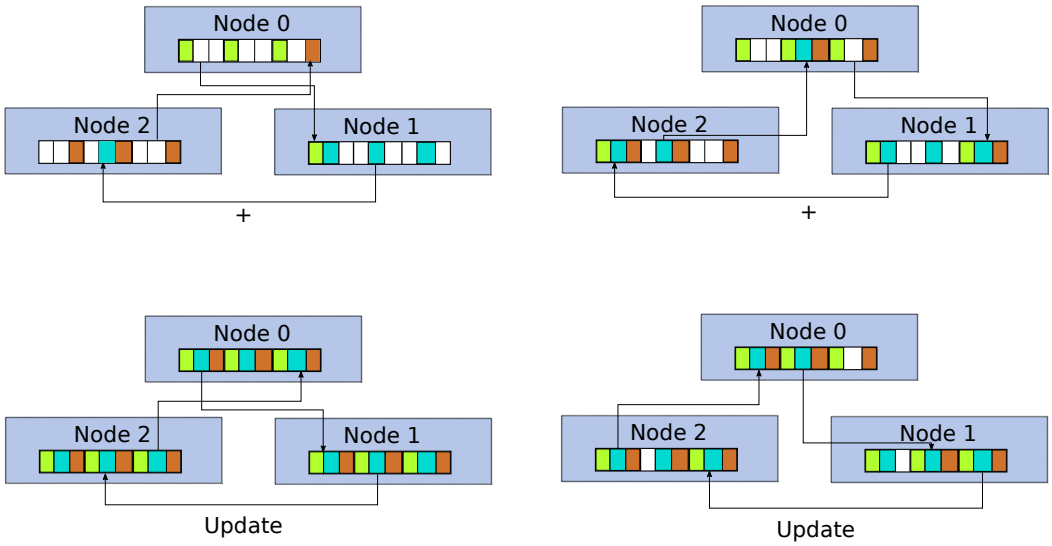


Figure 1.8 Horovod ring-allreduce implementation which communicates data between nodes or devices. The image show $M = 3$ nodes. First node communicates data in green, second node data in blue and third node data in orange.

Frameworks provide a ease of use implementations for deep learning algorithms. For instance, Caffe [34] provides a single and multi-GPU training for a single machine focusing in a group of initial users. On the other hand, Apache SINGA [86, 55] provides scalable deep learning for distributed environments. Furthermore, TensorFlow includes the scaling for CPU and GPU machines, cluster and mobile devices [1]. Also, Horovod [71] permits to uniformly distribute workload between multiple devices with an optimized ring all-reduce communication collective (which is shown in Figure 1.8), which improves the communication between processes. The procedure of the ring all-reduce collective is as follows. Each device

$(0 \ldots M)$ communicates with its neighbors $2 \times (M-1)$ times, where M is the number of nodes or devices. In the first $M-1$ iterations the values contained in the buffer of the receiving node are accumulated with the received ones. In each iteration, values in each node are calculated as the sum of the communicated values with the local node value. Subsequently, nodes updates the global values of the previous step locally. After that, each node have the complete global values. Another interesting framework is MXNet [14], which uses a communication structure named called KVStore with a two-level structure. The first level synchronizes data between the devices running on the same node. The second level is responsible for the communication and synchronization between different nodes. Finally, the PyTorch framework [60] provides versatility, simplicity and functionality to develop a wide variety of deep learning algorithms within distributed environments. In this context, PyTorch implements two different communication mechanisms. In this regard, it supports Remote Procedure Call (RPC) communications and three backends (named Gloo, NCCL and MPI), providing the communication primitives for multi-process parallelism. There are different reasons in the selection of a specific backend in PyTorch. For instance, NCCL provides the best GPU performance for Infiniband and Ethernet networks due to the optimized collective operations. On the other hand, MPI gains potential when a heterogeneous system is used, combining CPU and GPU, whilst Gloo works similar as NCCL with a lower performance. Related to this, Asaadi *et al.* [4] have conducted an interesting study about the communication issue considering heterogeneous environments. Also, there are several works that benchmark the performance of frameworks with respect to the training time [5, 76].

1.5.2. Workload balancing for deep learning image processing

The optimization of deep learning algorithms is performed by distributing the workload between the available processes. This can be defined as a hard NP-Problem. In order to solve this problem, different types of data processing parallelism have been proposed:

- *Data parallelism* is used to replicate the neural model among all available resources that compose the platform. Every resource holds an entire copy of the model and performs its own training procedure. In this sense, the data is shuffled and partitioned between the resources at each epoch so every training step is different in all replicas as exposed in Le *et al.* [48]. After the training step, replicas should communicate the obtained results using communication collectives. A popular example of communication collective is ring all-reduce [71].

- *Model parallelism* is situationally used when the model is too big to fit in the memory of the resources. In this context, every resource train its own part of the model with the

same portion of the data to later join the results and continue the training procedure. As training is inherently sequential, this should be done in specific situations and with a deep knowledge of the functionalities.

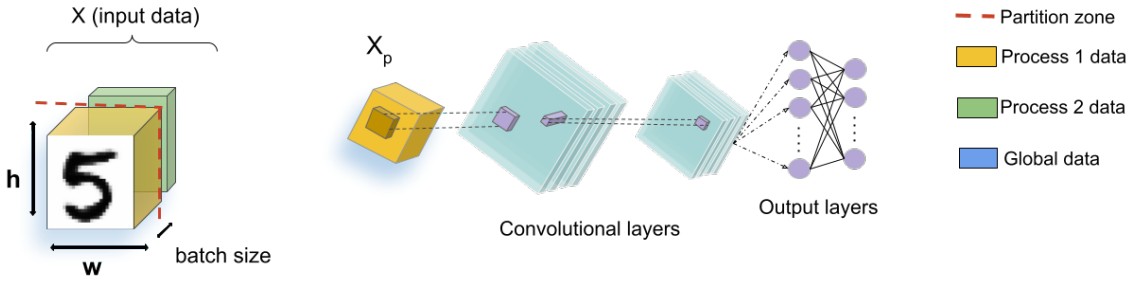
- *Hybrid parallelism* is the combination of the data and model parallelism approaches. This method is not common in the literature and it is only used under certain circumstances where there is a problem composed of an extremely big data amount and solved by a high complex model.

Nevertheless, it is worth noting that processes with different speeds are involved in the training of the deep learning models within heterogeneous platforms. Synchronization points imposed in the communication of intermediate results cause straggler processes and hence, affecting directly the training step by decaying the performance and increasing the total run-time. This is caused by the different computational capabilities of the devices, which leads to some processes (the faster ones) waiting for the rest and producing waiting times at synchronization points. In order to deal with this drawback, Visnhu *et al.* [84] proposed an implementation based on MPI to scale TensorFlow applications in HPC clusters. In this work, an evaluation of the scalability of this proposal is done by calculating the speedup of different schemes under a homogeneous partition of some deep learning datasets as MNIST [23] and CIFAR10 [39]. On the other hand, Dipankar *et al.* [21] estimated the computation and communication and proposed a distributed multi-node synchronous called Synchronous Stochastic Gradient Descent (SSGD). Focusing in data parallelism and assuming a perfect homogeneous platform, SSGD fits perfectly. Conversely, in heterogeneous platforms, faster processes should wait at the synchronization points. A depth study of the homogeneous parallelization methods is conducted by Bennun *et al.* [8]. In this work, an asynchronous stochastic optimization method using Asynchronous Stochastic Gradient Descent (ASGD) is proposed. This method solves the problem of synchronization points since replicas do not need to wait for the slower ones. Nevertheless, it adds a new problem called *staleness*. This happens when the training in each replica is computed using an old version of the parameters, and hence, causing a negative impact on the resulting model accuracy. Thus, a solution should be provided. Some works tried to solve the workload imbalance in heterogeneous platforms. For instance, Chen *et al.* [12] proposed a semi-dynamic load balancing to deal and reduce the stragglers processes of distributed ML workloads. The basic concept of this work is to have static workload within each iteration for each process but dynamic workload across different iterations. It implements a Load-Balanced Bulk Synchronous Parallel (LB-BSP) to equalize all devices batch processing times by re-configuring their batch sizes at the synchronization points. To do this, the goal is to find the device batch size $B = \{b_0, b_1, \ldots, b_p\}$ that minimize

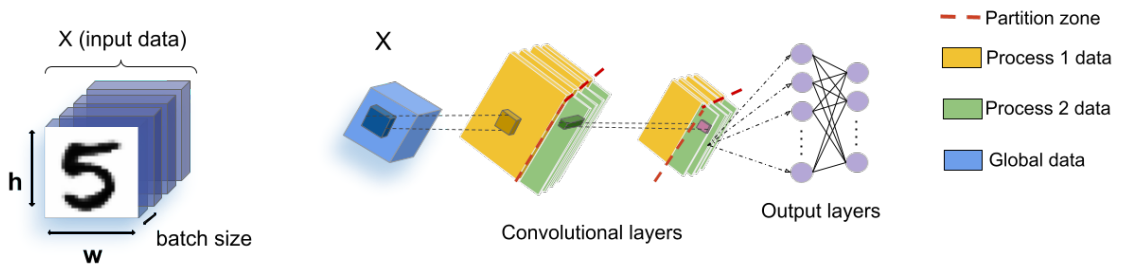
the processing time of the complete execution. The corresponding speed assigned to each process is calculated as the consumed time to complete the processing of input data x. Then, gradient aggregation is computed using the worker batch size when aggregating gradients following Equation (1.4)

$$g = \frac{1}{\sum_{p=0}^{P} |b_p|} \sum_{p=0}^{P} |b_p| \cdot g_p, \tag{1.4}$$

where $|b_p|$ is the process batch size, $X = \sum_{p=0}^{P} |b_p|$ is the total batch size and g_p the gradients of the p process. Finally, in the next iteration the new batch sizes values are dynamically re-calculated by following previous calculations again. Data parallelism functioning can be appreciated in the Figure 1.9(a) for $P = 2$ where the red line splits the input data in two uneven partitions x_p using the process speed.



a) Heterogeneous Data Parallelism scheme



b) Heterogeneous Model Parallelism scheme

Figure 1.9 Data and model parallelism for heterogeneous partitions considering CNN models. h, w define the spatial dimensions (height and width) of the data. Batch size is assigned to each process in terms of speed. The used image is from the MNIST data set, characterized by its size $(28, 28, 1)$.

Attending to model parallelism, the number of works is not high compared to data parallelism [77, 90, 75, 13]. In the forward pass of CNNs, the convolutional weight matrix is vertically partitioned and distributed over the available p resources for a specific layer of the neural model. This procedure is shown in the Figure 1.9(b). In the backward pass, processes use the error E and calculate the gradients for their corresponding subset using the output from the forward step. Finally, gradients are shared between processes.

1.6. Memory overload in deep learning data balancing

The amount of data used for conducting the training stage of any deep learning method can be extremely high and practically unmanageable. Thus, memory overload issues may appear causing severe limitations within the learning procedure. This drawback may occurs more frequently for large data such as hyper-spectral images, where the high spectral dimension and the large data variability severe hinder the accuracy of the deep model. Indeed, the way the data is stored and distributed in these applications can cause important memory problems. As a result, the model accuracy is also highly affected since the assigned training samples to each process is limited due to the high spectral-dimension of the data and the limitation of the partitions. Also, different stages in the execution of deep learning algorithms require the computation of an extremely big amount of floating points operations. Hence, a big amount of parameters should be stored in memory, which may require large memory spaces. In order to solve memory overload issues, different strategies have been proposed in the literature to address the balancing of data with a trade-off between the amount of data and the quality of the data [59, 57, 32]. The objective of these trade-off techniques is to obtain the features with more representation of the data while discriminating the rest of the features.

Chapter 2

Goals and motivation

In Chapter 1 we reviewed the challenges and issues related to workload partitioning in heterogeneous platforms and its implications in the computation and communication costs. A main conclusion is that the computational workload balance and the processes communication optimization have a decisive impact in the application overall execution time. One of the main objectives of this thesis is to apply previous described techniques and methods to improve performance of deep learning algorithms running on heterogeneous HPC platforms. In concrete, we address the following tasks:

1. Optimizing parallelism schemes for deep learning image classification while solving the *staleness* problem. The partitioning and distribution of the training data should be achieved using properties that characterize the different processes.

2. Developing methods to avoid memory overload issues and fault-tolerant techniques in distributed environments.

It is worth noting that this thesis stems as the continuation of the final master thesis work published in [64]. Furthermore, in the context of the master work, three contributions were published in scientific journals. These contributions are the followings:

1. Rico-Gallego, J. A., Diaz-Martin, J. C., Calvo-Jurado, C., **Moreno-Alvarez, S.** and Garcia-Zapata, J. L. (2019). Analytical Communication Performance Models as a metric in the partitioning of data-parallel kernels on heterogeneous platforms. The Journal of Supercomputing, 75(3), 1654-1669. [IF(2019)=2.469].

2. Rico-Gallego, J. A., Diaz-Martin, J. C., **Moreno-Alvarez, S.**, Calvo-Jurado, C. and Garcia-Zapata, J. L. (2020). Performance evaluation of model-driven partitioning algorithms for data-parallel kernels on heterogeneous platforms. Computational and Mathematical Methods, 2(1), e1017.

3. Rico-Gallego, J. A., **Moreno-Alvarez, S.**, Diaz-Martin, J. C. and Lastovetsky, A. L. (2020). A tool to assess the communication cost of parallel kernels on heterogeneous platforms. The Journal of Supercomputing, 76(6), 4629-4644. [IF(2019)=2.469].

Former contributions during this thesis period provide an extended description of each technique and methodology. They demonstrated the advantages of the proposed methods compared to other well-known literature methods. In the following Sections, the main contributions of this thesis are provided.

2.1. Optimizing deep learning data-parallelism in HPC.

In this Section, a brief summary of the first contribution of this thesis is provided. In that work, a heterogeneous load balancing technique for deep learning applications using the data-parallelism approach is proposed. The objective of the proposed work is to optimize models training performance in terms of time and model accuracy.

Deep learning data-parallelism approaches distribute the input data between processes, known as *replicas*. Our proposal is to perform a heterogeneous data partitioning and distribution between replicas running in a HPC platform. The data partitioning is performed using the FuPerMod tool [17], which models the computational capabilities of each replica in the platform. We provide a representative benchmark of the computations as input to FuPerMod, obtaining a meaningful speed profiling of each replica. We experimentally found that a simple neural network represents accurately the computations performed in the training of a deep learning model. FuPerMod returns a speed function for each replica $s_p(B)$, which varies through a range of batch sizes B. For each batch size, the speed obtained by FuPerMod is the inverse of the execution time.

The speed of each replica is determined by FuPerMod in a prior step to the training, assuming a dedicated platform, with no variations in the resources performance. Additionally, this method can be used in non-dedicated platforms by the application of fine-grain speed adaption during the training step [12]. Once the speed vector is obtained, the partitioning step assigns a batch size $|B_p|$ to each replica according to its speed. The sum of the batch sizes assigned to the replicas is the global batch size $\sum_{p=1}^{P} |B_p| = |B|$. Faster replicas are assigned with a larger batch size. As a consequence, each replica p trains its own copy of the model (see Figure 1.9(a)) with an non-overlapping input data subset of size $\frac{X}{B} \times B_p$. The objective of this speed-based partitioning is that all processes finish the computation of its assigned batch at the same time, and hence, minimize the waiting times in the communication step. This technique reduces the staleness problem, ensuring that each replica communicates up to

date gradients with other replicas, and hence, avoiding the training with staleness parameter values.

Replicas should communicate gradients in a prior step to perform parameter updates. Thus, global gradients are obtained using local gradients from each replicas. Global gradients are calculated with the information obtained from the different input data in each replica. The communication procedure is performed using parameter servers or communication collectives. Parameter servers use a centralized node to store gradients from each replica. Communication collectives are used as a decentralized method where each replica sends messages to the rest. In this proposal, the communication is done via Allreduce collective as $g^k = \frac{1}{P}\sum_{p=1}^{P} g_p^k$, being k the training iteration.

This work is based on the PyTorch framework and the implementation is hence compatible with centralized data interchange using parameter servers. Finally, the experimentation results show that this implementation maintains the accuracy of a deep model while decreases substantially the training times and avoids straggler processes and staleness. Results are described in the first paper of the compendium of publications of this thesis.

2.2. Convolutional layers partitioning in distributed model parallelism

In this Section, the second contribution of this thesis is briefly described. In this contribution, a heterogeneous load balancing technique for deep models using the model-parallelism approach is proposed. The objective of the proposal is to partition specific parts of the model that are computationally expensive to accelerate the training. Meanwhile, the memory of the processes is controlled to avoid overloads.

Deep learning models may suffer from communication bottlenecks and memory overloads due to the high amount of data and model parameters processed, which should be stored in memory. Current data partitioning solutions in the literature assume homogeneous resources, however, in current heterogeneous HPC platforms, memory constraints should be considered to avoid memory overloads. In this thesis, we address a heterogeneous partitioning over the convolutional layers of convolutional neural networks (CNNs).

Assuming a CNN, our method splits computing tensors across the spectral dimension (filters). Tensors are multidimensional arrays used in computations of deep learning models. Filters in convolutional networks are implemented as tensors. Following the heterogeneous partitioning we propose, each replica computes a specific number of filters, assigned based on its computational capabilities, and later combines results with the rest of replicas.

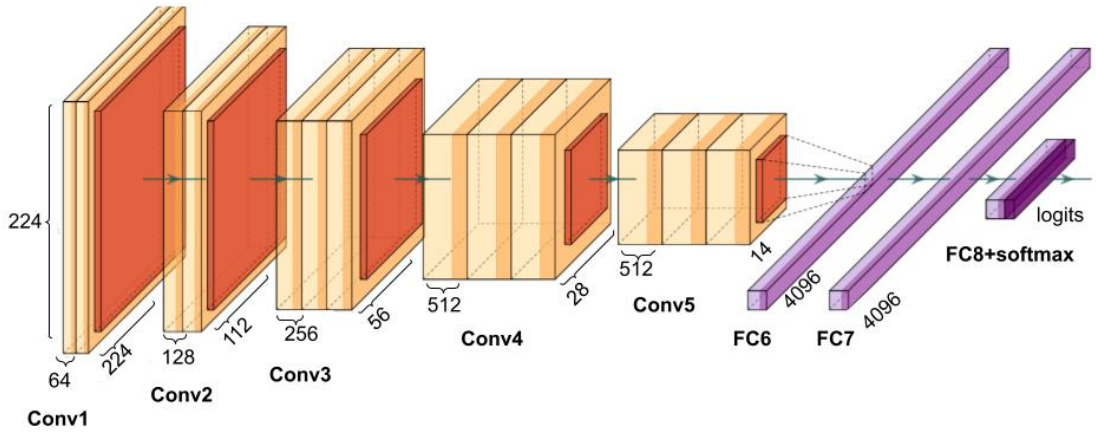


Figure 2.1 Example of a VGG-16 network for the heterogeneous model parallelism proposal.

The workload of each process is computed as (B, W_l, H_l, f_l^p), with p the process number, l the layer, W, H the width and height of the input data, f the filters in the layer and B the batch size. During these computations, tensors are computed partially or totally in each process, in case they are split or not. The partitioned tensors are communicated to each process using the allreduce collective. Thus, tensors are concatenated after each partitioned layer to compose the complete tensor. As a consequence, the total number of parameters of the model is defined as $\sum_{l=1}^{L} k_l \times B_l \times F_l$, where L is the total number of layers. Thus, heterogeneous partitioning is performed over the filter dimension $f_l^p = f_l \times s_p$, where $\sum_{p=1}^{P} f_l^p = F_l$.

The performance analysis have been carried out on deep learning models VGG11, VGG16, VGG19. An example of one of these networks is shown in Figure 2.1. Scientific data sets used to train the models are MNIST and CIFAR100. MNIST is composed handwritten digits images of size $28 \times 28 \times 1$, representing digits from 0 to 9, with a training set of 60.000 examples and a test set of 10.000 examples. CIFAR100 contains 60.000 colour images of size $32 \times 32 \times 3$ of 100 non-biased classes. The models are evaluated in a heterogeneous HPC platform with four NVIDIA Volta V100 and an Intel Xeon Gold 6240 CPU. Results show an improvement of the accuracy for evaluated models and a notable training acceleration. Meanwhile, memory overloads are avoided. Results are shown in the second article of the compendium of publications of this thesis.

2.3. Deep learning memory usage optimization

In this Section, the third contribution of this thesis is described. The experimentation is performed using remote sensing scenes. The objective of this proposal is to improve the memory usage in the training step while accelerating the execution.

Remote sensing images [42, 31] capture different wavelength channels with detailed information from the spatial and spectral dimensions of a scene. The resulting images have a large size, and the deep learning models used to process them contain a large amount of parameters. Hence, it is necessary to reduce the memory usage to avoid overload problems.

Most deep learning model architectures use single-precision values (FP32) for the computations in the training step. In order to optimize computations, two challenges appear. First, the memory should store FP32 operations, which is very costly. Secondly, computations should be reduced in order to speed up the training without affecting the deep network accuracy. Examples of the high cost of processing FP32 operations are Capsule Networks [57], Residual and Dense networks [59], which perform training and inference stages using single-precision floating operations with remote sensing scenes as input data. We propose to address both challenges to reduce the high computational demand of remote sensing images classification tasks in different NVIDIA GPUs architectures described in Table 2.1. Following, our proposal is described.

Model	**Architecture**	**GFLOPS FP32**	**GFLOPS FP16**	**Frequencie FP32**	**L2 Cache**	**L1 Cache**	**L0 Cache**
Tegra TK	Kepler	365	-	951	1536	48	-
Tegra X1	Maxwell	649	1298	1267	2048	32	-
Tegra X2	Pascal	750	1465	1267	4096	32	12
Xavier	Volta	1410	2820	1377	6144	256	12

Table 2.1 NVIDIA GPUs properties for half-precision (FP16) and single-precision (FP32). Frequency is shown in (Mhz) and cache size is shown in kilobytes (KBs).

Memory saving while decreasing computation can be achieved by casting specific operations from FP32 to half-precision FP16. The key point to do this transformations effectively is to prevent the model accuracy from dropping. Our proposal is to perform mixed precision by maintaining an accumulative copy of the weights in FP32 and performing mixed precision (combination of FP32 and FP16) operations during the whole training step. FP16 is used for the forward and backward propagation. Meanwhile, FP32 is used for the loss computation and weight updating. Some operations need to be represented in FP32 since the FP16 approximation does not provide good results. Meanwhile, some data as the weights and gradients are cast into FP16. This entails the problem of precision errors as zero gradients or incorrect values in loss and activations when the magnitudes of the values are lower than 2^{-24}. In order to solve that issue, an accumulative copy of the weights is used in the weight updating step. Additionally, it is necessary to perform loss scaling to push gradients values up to higher values and avoiding zero gradients.

We use the Apex Automatic Mixed Precision (AMP) [37] of PyTorch framework to implement this technique. AMP provides the functionalities to transform FP32 to and from FP16. Experiments are conducted over the Indian Pines (IP), Salinas Valley (SV), Kennedy Space Center (KSC) and University of Pavia (UP) hyperspectral images as input dataset and using multiple deep models. Results show a high improvement in the computation step time while maintaining accuracy. In addition, the memory consumption is significantly reduced. Results are shown in the third publication of the compendium of publications of this thesis.

2.4. Deep learning acceleration in non-dedicated environments

In this Section, a summary of the fourth publication of this thesis is described. In this proposal, deep learning models are trained in non-dedicated cloud platforms. The input of the deep learning models are remote sensing images. The objective of this proposal is to accelerate the training step in cloud computing environments. Meanwhile, a scaling study is provided to evaluate performance for increasing amount of data and processes.

Dedicated HPC platforms have previously been used as target environments. Since this thesis addresses the usage of heterogeneous platforms to train deep models, non-dedicated platforms as cloud computing should be considered. Cloud computing platforms are commonly used for the processing of massive data amounts in deep learning algorithms. The challenges of processing complex data over multiple replicas distributed across non-dedicated platforms are also present in cloud environments due its heterogeneous nature. Clouds integrate a high number of resources distributed in different locations and connected with different networks.

Computing intensive remote sensing images have not been usually processed using cloud computing approaches. However, there are some thorough works which take advantage of the distribution and acceleration of the data processing time for remote sensing scenes [87, 88, 92] offered by cloud computing. Cloud computing can significantly reduce the training time of deep networks, providing with a reduced cost hardware solution. In cloud computing, users pay by the usage time on the platforms while the platform provides with a set of expensive resources. An important additional benefit is the fault-tolerance support of this platforms. Fault-tolerance aims to maintain the training of the deep learning algorithm in case that a node fails, re-hosting the affected tasks in other nodes. This re-hosting is also useful for situational memory overload errors.

The experimentation carried out in this part of the work consists in performing computational load balancing of a fully connected deep neural network composed of 4 hidden layers with 144 neurons per layer. The output layer is composed of 58 neurons. The experimentation is divided in three parts. First, a scaling evaluation is performed by increasing the number of replicas. Results show that the speedup increases with the number of replicas, as expected. However, the huge data volume to process avoids to speedup the training without causing communication bottlenecks. Secondly, the training is accelerated for multiple data sizes. The acceleration is based on the data distribution. A study of the run-time increment depending of the model input data is performed. Finally, we compare the accuracy obtained with the PyTorch framework to our proposed method. Results are very promising and they are detailed in the fourth publication of the compendium of publications of this thesis.

Chapter 3

Conclusions

This thesis proposes and evaluates heterogeneous workload balancing techniques for scientific applications and deep learning algorithms in HPC and cloud platforms. Following, we summarize the steps in the implementations of each contribution.

In HPC scientific applications and kernels, we propose and evaluate the use of analytical models of the communications as a optimization objective. Previously, workload is distributed heterogeneously between processes according to their computing capabilities. Regarding deep learning algorithms, we propose to apply several optimization techniques to improve the performance of the training of deep models. These techniques should the accuracy of trained model and consider memory storage restrictions.

Following, we highlight a set of conclusions obtained from our contributions to the fields of discussion:

- Communication waiting times in distributed environments are critical to improve the applications performance. Current data partitioning techniques aims to distribute the computation based on a limited representation of the communications.
- Deep learning frameworks does not consider the heterogeneity of the resources in the platform. As a consequence, the processing time and accuracy of the model are highly impacted.
- The implementation of heterogeneous balancing techniques improves the training performance, however, they could produce memory overload issues. While non-dedicated cloud platforms solve this issues by re-hosting failed replicas, the usage of techniques as mixed-precision has been demonstrated useful to accelerate training time and avoiding memory overloads while maintaining the accuracy.

To conclude this summary, the different works described in this thesis have been published in several journals and conferences, to a total of six journal publications and four conference presentations. Some of them have been developed during the master period, previous to the doctoral work. Finally, it is worth noting that this thesis has coincided in time with the COVID-19 pandemic during almost all of its development, limiting the assistance to conferences during that time.

Bibliography

[1] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I. J., Harp, A., Irving, G., Isard, M., Jia, Y., Józefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D. G., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P. A., Vanhoucke, V., Vasudevan, V., Viégas, F. B., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2016). Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *ArXiv*, abs/1603.04467.

[2] Agosta, G., Fornaciari, W., Massari, G., Pupykina, A., Reghenzani, F., and Zanella, M. (2018). Managing heterogeneous resources in hpc systems. In *Proceedings of the 9th Workshop and 7th Workshop on Parallel Programming and RunTime Management Techniques for Manycore Architectures and Design Tools and Architectures for Multicore Embedded Computing Platforms*, PARMA-DITAM '18, page 7–12, New York, NY, USA. Association for Computing Machinery.

[3] Albawi, S., Mohammed, T. A., and Al-Zawi, S. (2017). Understanding of a convolutional neural network. In *2017 International Conference on Engineering and Technology (ICET)*, pages 1–6. IEEE.

[4] Asaadi, H. and Chapman, B. (2017). Comparative study of deep learning framework in hpc environments. In *2017 New York Scientific Data Summit (NYSDS)*, pages 1–7.

[5] Bahrampour, S., Ramakrishnan, N., Schott, L., and Shah, M. (2015). Comparative study of caffe, neon, theano, and torch for deep learning. *ArXiv*, abs/1511.06435.

[6] Beaumont, O., Becker, B. A., DeFlumere, A., Eyraud-Dubois, L., Lambert, T., and Lastovetsky, A. (2019). Recent advances in matrix partitioning for parallel computing on heterogeneous platforms. *IEEE Transactions on Parallel and Distributed Systems*, 30(1):218–229.

[7] Beaumont, O., Boudet, V., Rastello, F., and Robert, Y. (2001). Matrix multiplication on heterogeneous platforms. *IEEE Transactions on Parallel and Distributed Systems*, 12(10):1033–1051.

[8] Ben-Nun, T. and Hoefler, T. (2018). Demystifying parallel and distributed deep learning: An in-depth concurrency analysis. *arXiv: Learning*.

[9] Bosque, J. and Pastor, L. (2006). A parallel computational model for heterogeneous clusters. *IEEE Transactions on Parallel and Distributed Systems*, 17(12):1390–1400.

[10] Bosque, J. and Perez, L. (2004). Hloggp: a new parallel computational model for heterogeneous clusters. In *IEEE International Symposium on Cluster Computing and the Grid, 2004. CCGrid 2004.*, pages 403–410.

[11] Buryak, D., Popova, N., Voevodin, V., Konkov, Y., Ivanov, O., Shaykhlislamov, D., and Fateev, I. (2019). An experimental study of deep neural networks on hpc clusters. In Voevodin, V. and Sobolev, S., editors, *Supercomputing*, pages 489–504, Cham. Springer International Publishing.

[12] Chen, C., Weng, Q., Wang, W., Li, B., and Li, B. (2020). Semi-dynamic load balancing: efficient distributed learning in non-dedicated environments. *Proceedings of the 11th ACM Symposium on Cloud Computing*.

[13] Chen, C.-C., Yang, C.-L., and Cheng, H.-Y. (2018). Efficient and robust parallel dnn training through model parallelism on multi-gpu platform. *ArXiv*, abs/1809.02839.

[14] Chen, T., Li, M., Li, Y., Lin, M., Wang, N., Wang, M., Xiao, T., Xu, B., Zhang, C., and Zhang, Z. (2015). Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. *CoRR*, abs/1512.01274.

[15] Choquette, J., Gandhi, W., Giroux, O., Stam, N., and Krashinsky, R. (2021). Nvidia a100 tensor core gpu: Performance and innovation. *IEEE Micro*, 41(2):29–35.

[16] Chunduri, S., Groves, T., Mendygral, P., Austin, B., Balma, J., Kandalla, K., Kumaran, K., Lockwood, G., Parker, S., Warren, S., Wichmann, N., and Wright, N. (2019). Gpcnet: Designing a benchmark suite for inducing and measuring contention in hpc networks. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '19, New York, NY, USA. Association for Computing Machinery.

[17] Clarke, D., Zhong, Z., Rychkov, V., and Lastovetsky, A. (2014). Fupermod: A software tool for the optimization of data-parallel applications on heterogeneous platforms. *J. Supercomput.*, 69(1):61–69.

[18] Coates, A., Huval, B., Wang, T., Wu, D., Catanzaro, B., and Andrew, N. (2013). Deep learning with cots hpc systems. In Dasgupta, S. and McAllester, D., editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 1337–1345, Atlanta, Georgia, USA. PMLR.

[19] Costanzo, M., Rucci, E., Costi, U., Chichizola, F., and Naiouf, M. (2021). Comparison of hpc architectures for computing all-pairs shortest paths. intel xeon phi knl vs nvidia pascal. In Pesado, P. and Eterovic, J., editors, *Computer Science – CACIC 2020*, pages 37–49, Cham. Springer International Publishing.

[20] Czarnul, P. and Rościszewski, P. (2014). Optimization of execution time under power consumption constraints in a heterogeneous parallel system with gpus and cpus. In Chatterjee, M., Cao, J.-n., Kothapalli, K., and Rajsbaum, S., editors, *Distributed Computing and Networking*, pages 66–80, Berlin, Heidelberg. Springer Berlin Heidelberg.

[21] Das, D., Avancha, S., Mudigere, D., Vaidyanathan, K., Sridharan, S., Kalamkar, D. D., Kaul, B., and Dubey, P. (2016). Distributed deep learning using synchronous stochastic gradient descent. *CoRR*, abs/1602.06709.

[22] Dean, J., Corrado, G., Monga, R., Chen, K., Devin, M., Mao, M., Ranzato, M. a., Senior, A., Tucker, P., Yang, K., Le, Q., and Ng, A. (2012). Large scale distributed deep networks. In Pereira, F., Burges, C. J. C., Bottou, L., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc.

[23] Deng, L. (2012). The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142.

[24] Dichev, K. and Lastovetsky, A. (2014). Optimization of collective communication for heterogeneous hpc platforms. *High-Performance Computing on Complex Environments*, pages 95–114.

[25] Dieleman, S., Willett, K., and Dambre, J. (2015). Rotation-invariant convolutional neural networks for galaxy morphology prediction. *Monthly Notices of the Royal Astronomical Society*, 450.

[26] Ding, N., Xu, S., Song, Z., Zhang, B., Li, J., and Zheng, Z. (2019). Using hardware counter-based performance model to diagnose scaling issues of hpc applications. *Neural Computing and Applications*, 31(5):1563–1575.

[27] Dongarra, J., Luszczek, P., and Padua, D. (2011). *TOP500*, pages 2055–2057. Springer US, Boston, MA.

[28] Flich, J., Hernandez, C., Quiñones, E., and Paredes, R. (2020). Distributed training on a highly heterogeneous hpc system. In Orailoglu, A., Jung, M., and Reichenbach, M., editors, *Embedded Computer Systems: Architectures, Modeling, and Simulation*, pages 359–370, Cham. Springer International Publishing.

[29] Fox, G., Glazier, J., Kadupitiya, J., Jadhao, V., Kim, M., Qiu, J., Sluka, J. P., Somogyi, E., Marathe, M., Adiga, A., Chen, J., Beckstein, O., and Jha, S. (2019). Learning everywhere: Pervasive machine learning for effective high-performance computation. *2019 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 422–429.

[30] Goyal, N., Balasubramaniam, S., Goyal, P., Islam, S., and Sati, M. (2016). A high performance computing framework for data mining. In *2016 IEEE 23rd International Conference on High Performance Computing Workshops (HiPCW)*, pages 11–18.

[31] Green, R. O., Eastwood, M. L., Sarture, C. M., Chrien, T. G., Aronsson, M., Chippendale, B. J., Faust, J. A., Pavri, B. E., Chovit, C., Solis, M., Olah, M. R., and Williams, O. (1998). Imaging spectroscopy and the airborne visible/infrared imaging spectrometer (aviris). *Remote Sensing of Environment*, 65:227–248.

[32] Haut, J. M., Paoletti, M. E., Plaza, J., Li, J. Y., and Plaza, A. J. (2018). Active learning with convolutional neural networks for hyperspectral image classification using a new bayesian approach. *IEEE Transactions on Geoscience and Remote Sensing*, 56:6440–6461.

[33] Hockney, R. W. (1994). The communication challenge for mpp: Intel paragon and meiko cs-2. *Parallel Computing*, 20(3):389–398.

[34] Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S., and Darrell, T. (2014). Caffe: Convolutional architecture for fast feature embedding. *MM 2014 - Proceedings of the 2014 ACM Conference on Multimedia*.

[35] Kesselheim, S., Herten, A., Krajsek, K., Ebert, J., Jitsev, J., Cherti, M., Langguth, M., Gong, B., Stadtler, S., Mozaffari, A., Cavallaro, G., Sedona, R., Schug, A., Strube, A., Kamath, R., Schultz, M. G., Riedel, M., and Lippert, T. (2021). Juwels booster – a supercomputer for large-scale ai research.

[36] Khaleghzadeh, H., Manumachu, R. R., and Lastovetsky, A. (2018). A novel data-partitioning algorithm for performance optimization of data-parallel applications on heterogeneous hpc platforms. *IEEE Transactions on Parallel and Distributed Systems*, 29(10):2176–2190.

[37] Kim, D., Kwon, Y., Liu, P., Kim, I. L., Perry, D. M., Zhang, X., and Rodriguez-Rivera, G. (2016). Apex: Automatic programming assignment error explanation. *SIGPLAN Not.*, 51(10):311–327.

[38] Klinkenberg, J., Terboven, C., Lankes, S., and Müller, M. S. (2017). Data mining-based analysis of hpc center operations. In *2017 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 766–773. IEEE.

[39] Krizhevsky, A., Nair, V., and Hinton, G. (2010). Cifar-10 (canadian institute for advanced research). *URL http://www. cs. toronto. edu/kriz/cifar. html*, 5:4.

[40] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In Pereira, F., Burges, C. J. C., Bottou, L., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc.

[41] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2017). Imagenet classification with deep convolutional neural networks. *Commun. ACM*, 60(6):84–90.

[42] Kunkel, B., Blechinger, F., Lutz, R., Doerffer, R., van der Piepen, H., and Schroder, M. (1988). ROSIS (Reflective Optics System Imaging Spectrometer) - A Candidate Instrument For Polar Platform Missions. In Bowyer, C. S. and Seeley, J. S., editors, *Optoelectronic Technologies for Remote Sensing from Space*, volume 0868, pages 134 – 141. International Society for Optics and Photonics, SPIE.

[43] Kwak, G. H.-J. and Hui, P. (2019). Deephealth: Deep learning for health informatics. *ACM Transactions on Computing for Healthcare*.

[44] Kwon, Y. and Rhu, M. (2018). A case for memory-centric hpc system architecture for training deep neural networks. *IEEE Computer Architecture Letters*, 17(2):134–138.

[45] Lastovetsky, A., Mkwawa, I.-H., and O'Flynn, M. (2006). An accurate communication model of a heterogeneous cluster based on a switch-enabled ethernet network. In *12th International Conference on Parallel and Distributed Systems - (ICPADS'06)*, volume 2, pages 6 pp.–.

[46] Lastovetsky, A. and Reddy, R. (2007). Data Distribution for Dense Factorization on Computers with Memory Heterogeneity. *Parallel Comput.*, 33(12):757–779.

[47] Le, Q., Ranzato, M., Monga, R., Devin, M., Chen, K., Corrado, G., Dean, J., and Ng, A. (2011a). Building high-level features using large scale unsupervised learning. *Proceedings of ICML*, 1.

[48] Le, Q. V., Ngiam, J., Coates, A., Lahiri, A., Prochnow, B., and Ng, A. Y. (2011b). On optimization methods for deep learning. In *Proceedings of the 28th International Conference on International Conference on Machine Learning*, ICML'11, pages 265–272, USA. Omnipress.

[49] LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature*, 521(7553):436–444.

[50] LeCun, Y. A. (2019). 1.1 deep learning hardware: Past, present, and future. *2019 IEEE International Solid- State Circuits Conference - (ISSCC)*, pages 12–19.

[51] Leung, C. K., Sarumi, O. A., and Zhang, C. Y. (2020). Predictive analytics on genomic data with high-performance computing. In *2020 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, pages 2187–2194. IEEE.

[52] Marshall, J., Rickard, D., Sova, D., Miller, H., Lapihuska, R., Dennis, A., and Graziano, M. (2017). Heterogeneous high performance computing modules for next generation onboard processing. In *2017 IEEE Aerospace Conference*, pages 1–10.

[53] Mercier, G. and Jeannot, E. (2011). Improving mpi applications performance on multicore clusters with rank reordering. In *European MPI Users' Group Meeting*, pages 39–49. Springer.

[54] Milojicic, D., Faraboschi, P., Dube, N., and Roweth, D. (2021). Future of hpc: Diversifying heterogeneity. In *2021 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 276–281.

[55] Ooi, B. C., Tan, K.-L., Wang, S., Wang, W., Cai, Q., Chen, G., Gao, J., Luo, Z., Tung, A. K. H., Wang, Y., Xie, Z., Zhang, M., and Zheng, K. (2015). Singa: A distributed deep learning platform. *Proceedings of the 23rd ACM international conference on Multimedia*.

[56] O'Shea, K. and Nash, R. (2015). An introduction to convolutional neural networks. *CoRR*, abs/1511.08458.

[57] Paoletti, M., Haut, J., Fernandez-Beltran, R., Plaza, J., Plaza, A., Li, J., and Pla, F. (2018). Capsule networks for hyperspectral image classification. *IEEE Transactions on Geoscience and Remote Sensing*, PP:1–16.

[58] Paoletti, M., Haut, J., Plaza, J., and Plaza, A. (2019a). Deep learning classifiers for hyperspectral imaging: A review. *ISPRS Journal of Photogrammetry and Remote Sensing*, 158:279–317.

[59] Paoletti, M. E., Haut, J. M., Fernández-Beltran, R., Plaza, J., Plaza, A. J., and Pla, F. (2019b). Deep pyramidal residual networks for spectral–spatial hyperspectral image classification. *IEEE Transactions on Geoscience and Remote Sensing*, 57:740–754.

[60] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R., editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc.

[61] Pham, T.-T., Pister, M., and Couvée, P. (2019). Recurrent neural network for classifying of hpc applications. *2019 Spring Simulation Conference (SpringSim)*, pages 1–12.

[62] Raina, R., Madhavan, A., and Ng, A. Y. (2009). Large-scale deep unsupervised learning using graphics processors. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, page 873–880, New York, NY, USA. Association for Computing Machinery.

[63] Ramirez-Gargallo, G., Garcia-Gasulla, M., and Mantovani, F. (2019). Tensorflow on state-of-the-art hpc clusters: a machine learning use case. In *2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, pages 526–533. IEEE.

[64] Rico-Gallego, J., Martín, J., Calvo-Jurado, C., Moreno-Álvarez, S., and Zapata, J. L. (2019). Analytical communication performance models as a metric in the partitioning of data-parallel kernels on heterogeneous platforms. *The Journal of Supercomputing*, 75.

[65] Rico-Gallego, J.-A. and Díaz-Martín, J.-C. (2015). tau-lop: Modeling performance of shared memory mpi. *Parallel Computing*, 46:14–31.

[66] Rico-Gallego, J.-A., Díaz-Martín, J.-C., and Lastovetsky, A. L. (2016). Extending t-lop to model concurrent mpi communications in multicore clusters. *Future Generation Computer Systems*, 61:66–82.

[67] Rico-Gallego, J.-A., Lastovetsky, A. L., and Díaz-Martín, J.-C. (2017). Model-based estimation of the communication cost of hybrid data-parallel applications on heterogeneous clusters. *IEEE Transactions on Parallel and Distributed Systems*, 28(11):3215–3228.

[68] Rościszewski, P. (2018). Optimization of hybrid parallel application execution in heterogeneous high performance computing systems considering execution time and power consumption.

[69] Sedona, R., Cavallaro, G., Jitsev, J., Strube, A., Riedel, M., and Benediktsson, J. A. (2019a). Remote sensing big data classification with high performance distributed deep learning. *Remote Sensing*, 11(24).

[70] Sedona, R., Cavallaro, G., Jitsev, J., Strube, A., Riedel, M., and Benediktsson, J. A. (2019b). Remote sensing big data classification with high performance distributed deep learning. *Remote Sensing*, 11(24).

[71] Sergeev, A. and Balso, M. D. (2018). Horovod: fast and easy distributed deep learning in tensorflow. *CoRR*, abs/1802.05799.

[72] Serizel, R. and Giuliani, D. (2014). Deep neural network adaptation for children's and adults' speech recognition. *Deep neural network adaptation for children's and adults' speech recognition*, pages 344–348.

[73] Shams, S., Platania, R., Lee, K., and Park, S.-J. (2017). Evaluation of deep learning frameworks over different hpc architectures. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, pages 1389–1396.

[74] Shanbhag, A., Madden, S., and Yu, X. (2020). A study of the fundamental performance characteristics of gpus and cpus for database analytics. In *Proceedings of the 2020 ACM SIGMOD international conference on Management of data*, pages 1617–1632.

[75] Shazeer, N. M., Cheng, Y., Parmar, N., Tran, D., Vaswani, A., Koanantakool, P., Hawkins, P., Lee, H., Hong, M., Young, C., Sepassi, R., and Hechtman, B. A. (2018). Mesh-tensorflow: Deep learning for supercomputers. *ArXiv*, abs/1811.02084.

[76] Shi, S. and Chu, X. (2018). Performance modeling and evaluation of distributed deep learning frameworks on gpus. *2018 IEEE 16th Intl Conf on Dependable, Autonomic and Secure Computing, 16th Intl Conf on Pervasive Intelligence and Computing, 4th Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress(DASC/PiCom/DataCom/CyberSciTech)*, pages 949–957.

[77] Shrivastava, D., Chaudhury, S., and Jayadeva (2017). A data and model-parallel, distributed and scalable framework for training of deep networks in apache spark. *ArXiv*, abs/1708.05840.

[78] Sorkunlu, N., Luong, D., and Chandola, V. (2018). dynamicmf: A matrix factorization approach to monitor resource usage in high performance computing systems. *2018 IEEE International Conference on Big Data (Big Data)*, pages 1302–1307.

[79] Sorokin, A., Malkovsky, S., Tsoy, G., Zatsarinnyy, A., and Volovich, K. (2020). Comparative performance evaluation of modern heterogeneous high-performance computing systems cpus. *Electronics*, 9(6).

[80] Sridharan, S., Vaidyanathan, K., Kalamkar, D., Das, D., Smorkalov, M. E., Shiryaev, M., Mudigere, D., Mellempudi, N., Avancha, S., Kaul, B., et al. (2018). On scale-out deep learning training for cloud and hpc. *arXiv preprint arXiv:1801.08030*.

[81] Sukharev, P. V., Vasilyev, N. P., Rovnyagin, M. M., and Durnov, M. A. (2017). Benchmarking of high performance computing clusters with heterogeneous cpu/gpu architecture. In *2017 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus)*, pages 574–577.

[82] Tang, X. and Fu, Z. (2020). Cpu–gpu utilization aware energy-efficient scheduling algorithm on heterogeneous computing systems. *IEEE Access*, 8:58948–58958.

[83] van de Geijn, R. A. and Watts, J. (1997). Summa: scalable universal matrix multiplication algorithm. *Concurr. Pract. Exp.*, 9:255–274.

[84] Vishnu, A., Siegel, C., and Daily, J. (2016). Distributed tensorflow with mpi. *ArXiv*, abs/1603.02339.

[85] Wang, C. (2017). High performance computing for big data: Methodologies and applications.

[86] Wang, W., Chen, G., Dinh, T. T. A., Gao, J., Ooi, B. C., Tan, K.-L., and Wang, S. (2015). Singa: Putting deep learning in the hands of multimedia users. *Proceedings of the 23rd ACM international conference on Multimedia*.

[87] Wu, Z., Sun, J., Zhang, Y., Zhu, Y., Li, J., Plaza, A., Benediktsson, J. A., and Wei, Z. (2021). Scheduling-guided automatic processing of massive hyperspectral image classification on cloud computing architectures. *IEEE Transactions on Cybernetics*, 51(7):3588–3601.

[88] Yao, X., Li, G., Xia, J., Ben, J., Cao, Q., Zhao, L., Ma, Y., Zhang, L., and Zhu, D. (2020). Enabling the big earth observation data via cloud computing and dggs: Opportunities and challenges. *Remote. Sens.*, 12:62.

[89] Young, T., Hazarika, D., Poria, S., and Cambria, E. (2018). Recent trends in deep learning based natural language processing. *ieee Computational intelligenCe magazine*, 13(3):55–75.

[90] Zaharia, M., Xin, R. S., Wendell, P., Das, T., Armbrust, M., Dave, A., Meng, X., Rosen, J., Venkataraman, S., Franklin, M. J., Ghodsi, A., Gonzalez, J., Shenker, S., and Stoica, I. (2016). Apache spark: A unified engine for big data processing. *Commun. ACM*, 59(11):56–65.

[91] Zahn, F. and Fröning, H. (2020). On network locality in mpi-based hpc applications. In *49th International Conference on Parallel Processing-ICPP*, pages 1–10.

[92] Zheng, P., Wu, Z., Sun, J., Zhang, Y., Zhu, Y., Shen, Y., Yang, J., Wei, Z., and Plaza, A. (2021). A parallel unmixing-based content retrieval system for distributed hyperspectral imagery repository on cloud computing platforms. *Remote Sensing*, 13:176.

[93] Zheng, W. (2020). Research trend of large-scale supercomputers and applications from the top500 and gordon bell prize. *Science China Information Sciences*, 63:1–14.

[94] Zhou, J. and Troyanskaya, O. (2014). Deep supervised and convolutional generative stochastic network for protein secondary structure prediction. In Xing, E. P. and Jebara, T., editors, *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, pages 745–753, Bejing, China. PMLR.

Thesis publications

The following publications have been achieved in the context of this thesis work.

Journal Papers:

1. **Moreno-Alvarez, S.**, Haut, J. M., Paoletti, M. E., Rico-Gallego, J. A., Diaz-Martin, J. C. and Plaza, J. (2020). Training deep neural networks: a static load balancing approach. The Journal of Supercomputing, 76(12), 9739-9754. [IF(2020)=2.474].

2. **Moreno-Alvarez, S.**, Haut, J. M., Paoletti, M. E. and Rico-Gallego, J. A. (2021). Heterogeneous model parallelism for deep neural networks. Neurocomputing, 441, 1-12. [IF(2020)=5.719].

3. Paoletti, M. E., Tao, X., Haut, J. M., **Moreno-Alvarez, S.** and Plaza, A. (2021). Deep mixed precision for hyperspectral image classification. The Journal of Supercomputing, 1-12. [IF(2020)=2.474].

4. Haut, J. M., Paoletti, M. E., **Moreno-Álvarez, S.**, Plaza, J., Rico-Gallego, J. A. and Plaza, A. (2021). Distributed Deep Learning for Remote Sensing Data Interpretation. Proceedings of the IEEE. [IF(2020)=10.961].

Submited Journal Papers:

1. **Moreno-Alvarez, S.**, Paolett, M. E. and Haut, J. M. (2020). Multiple Attention-Guided Capsule Networks for Hyperspectral Image Classification. IEEE Transactions on Geoscience and Remote Sensing. [IF(2020)=5.600].

Peer-Reviewed International Conference Papers:

1. **Moreno-Alvarez, S.**, Rico-Gallego, J. A., Haut, J. M., García Zapata, J. L. and Diaz-Martin, J. C. (2021). Mapping Optimization for HPC Infrastructures based on Reinforcement Learning. Proceedings of the 21th International Conference on Computational and Mathematical Methods in Science and Engineering (CMMSE'21), Rota, Spain, 2021.

Imparted Seminars:

1. **Moreno-Alvarez, Sergio., Juan A. Rico-Gallego, Juan Carlos Díaz-Martín, Juan Luis Garía-Zapata** (2020). Particionamiento Heterogéneo del Cómputo en Aplicaciones de Computación de Alto Rendimiento. Red de Computación de Altas Prestaciones sobre Arquitecturas Paralelas Heterogéneas (CAPAP-H). Barcelona, Enero 2020.

National Journals:

1. **Moreno-Alvarez, Sergio.** (2019). Equilibrado del Entrenamiento Distribuido de Redes Neuronales Profundas en Plataformas Heterogéneas: Actas de (III) Jornadas Doctorales, Badajoz, 29 de Noviembre de 2019.

Copyright notice

Escuela Politecnica
Av. de la Universidad, S/N, 10003
Caceres, Spain
Phone: 0034927257000. Ext. 51655
Email: jarico,juanmariohaut{@unex.es}

Dr. Juan Antonio Rico Gallego y Dr. Juan Mario Haut Hurtado como directores de la tesis titulada "Desarrollo de técnicas de aprendizaje automático para la optimización de aplicaciones científicas y de procesamiento masivo de datos en entornos de altas prestaciones", acreditan la colaboración y aportación del doctorando en el trabajo. Asímismo, se certifica también el factor de impacto junto con la respectiva categoría de la siguiente publicación incorporada en la tesis doctoral. Para cualquier aclaración con respecto a lo indicado, por favor, contacten con cualquiera de los directores.

Juan Antonio Rico Gallego PhD and Juan Mario Haut Hurtado PhD as directors of the PhD thesis titled "Development of machine learning techniques for the optimization of scientific and massive data processing applications in high performance computing environments", certify the collaboration and contribution of the doctoral student at work. Likewise, the impact factor is also certified along with the respective category of the following publication incorporated in the doctoral thesis. For any clarification regarding what is indicated, please contact any of the directors.

Artículo / Paper

Authors: **S. Moreno-Álvarez**, J. M. Haut, M. E. Paoletti, J. A. Rico-Gallego, J. C. Diaz-Martin and J. Plaza.
Title: Training deep neural networks: a static load balancing approach.
Journal: Journal of Supercomputing.
Other Information: vol 76, pp 9739–9754, 2020.
DOI: 10.1007/s11227-020-03200-6.
Impact factor 2020: 2.474. Q2
Abstract: Deep neural networks are currently trained under data-parallel setups on high-performance computing (HPC) platforms, so that a replica of the full model is charged to each computational resource using non-overlapped subsets known as batches. Replicas combine the computed gradients to update their local copies at the end of each batch. However, differences in performance of resources assigned to replicas in current heterogeneous platforms induce waiting times when synchronously combining gradients, leading to an overall performance degradation. Albeit asynchronous communication of gradients has been proposed as an alternative, it suffers from the so-called staleness problem. This is due to the fact that the training in each replica is computed using a stale version of the parameters, which negatively impacts the accuracy of the resulting model. In this work, we study the application of well-known HPC static load balancing techniques to the distributed training of deep models. Our approach is assigning a different batch size to each replica, proportional to its relative computing capacity, hence minimizing the staleness problem. Our experimental results (obtained in the context of a remotely sensed hyperspectral image processing application) show that, while the classification accuracy is kept constant, the training time substantially decreases with respect to unbalanced training. This is illustrated using heterogeneous computing platforms, made up of CPUs and GPUs with different performance.

Contribución del doctorado: Planteamiento de la hipótesis, desarrollo práctico, análisis y discusión de los resultados, elaboración y escritura del manuscrito.

Firma / Signature
Nov / Nov, 2021

Juan Antonio Rico Gallego

Juan Mario Haut Hurtado

The Journal of Supercomputing (2020) 76:9739–9754
https://doi.org/10.1007/s11227-020-03200-6

Training deep neural networks: a static load balancing approach

Sergio Moreno-Álvarez[1] · Juan M. Haut[2] · Mercedes E. Paoletti[2] · Juan A. Rico-Gallego[1] · Juan C. Díaz-Martín[2] · Javier Plaza[2]

Published online: 2 March 2020

Abstract

Deep neural networks are currently trained under data-parallel setups on high-performance computing (HPC) platforms, so that a replica of the full model is charged to each computational resource using non-overlapped subsets known as batches. Replicas combine the computed gradients to update their local copies at the end of each batch. However, differences in performance of resources assigned to replicas in current heterogeneous platforms induce waiting times when synchronously combining gradients, leading to an overall performance degradation. Albeit asynchronous communication of gradients has been proposed as an alternative, it suffers from the so-called staleness problem. This is due to the fact that the training in each replica is computed using a stale version of the parameters, which negatively impacts the accuracy of the resulting model. In this work, we study the application of well-known HPC static load balancing techniques to the distributed training of deep models. Our approach is assigning a different batch size to each replica, proportional to its relative computing capacity, hence minimizing the staleness problem. Our experimental results (obtained in the context of a remotely sensed hyperspectral image processing application) show that, while the classification accuracy is kept constant, the training time substantially decreases with respect to unbalanced training. This is illustrated using heterogeneous computing platforms, made up of CPUs and GPUs with different performance.

Keywords Deep learning · High-performance computing · Distributed training · Heterogeneous platforms

✉ Sergio Moreno-Álvarez
smoreno@unex.es

Extended author information available on the last page of the article

1 Introduction

Deep learning (DL) algorithms based on neural network architectures [19] have reached great accuracy in areas such as image classification [17, 20] and speech recognition [5] among others. When compared with other machine learning (ML) and pattern recognition methods, deep neural networks (DNNs) work as universal approximators of parameterized maps (models) composed of stacks of layers [12], where each one is composed by several nodes (neurons) connected to the nodes of the precedent and subsequent layers through synaptic weights and saturation control biases [22]. Overall, DNN models fit neuron weights and biases through an iterative optimization process based on training with examples. Improvements with respect to traditional techniques are supported by the large amount of data available to train these models, as well as by advances in high-performance computing (HPC) platforms [9].

DNN learning strategies can be roughly classified into supervised and unsupervised learning [14], depending on whether they use labeled data or not. This work focuses on supervised image classification with DNNs, whose input dataset, $\mathcal{X}$, is composed of n_{examples} images of $\mathbf{x}_n \in \mathbb{R}^{h \times w \times c}$ $(n = 1, \ldots, n_{\text{examples}})$, where $h \times w$ denotes the spatial dimensions of the images (i.e., height and width) and c the spectral depth (i.e., the number of channels). $\mathcal{X}$ is divided into two main subsets. The first one is the so-called training set, on which the classifier adjusts its weights and biases. The second set is the test set, on which the classifier makes the inference. With this in mind, it is easy to describe the DNN for image classification as a mapping model $\mathcal{M}(\cdot, \theta)$ with parameter θ that performs $\mathcal{M} : \mathcal{X} \rightarrow \mathcal{Y}$, associating each image of the original dataset ($\mathcal{X}$) with a corresponding label ($\mathcal{Y}$) by adjusting the parameters of the model θ. On this wise, the purpose of supervised learning is to find optimal $\theta^\star$ values in order to minimize the distance between the outputs of the model and the labeled values. Such distance is determined by the so-called *loss function* $L(\theta)$ (e.g., mean square error) during the training stage. The *Stochastic Gradient Descent* (SGD) method is commonly used for this purpose. In each iteration k, SGD updates θ through gradient g^k of $L(\theta)$ as $\theta^{k+1} = \theta^k - \alpha\, g^k$, where α is the so-called *learning rate*, which controls the advance in the weight domain. g^k is computed as $g^k = \frac{1}{|\chi|} \sum_{n \in \chi} \nabla l(n|\theta^k)$, being $l(n|\theta^k)$ the loss of the example n, computed on θ^k .

The updating process demands high computational capacity. To accelerate it, dedicated HPC clusters and non-dedicated cloud platforms are commonly used for large-scale deep networks training on large datasets. The training process is usually distributed on the platform resources based on two main schemes of parallelization, known as *model parallelism* and *data parallelism*.

Model parallelism is used when the model is too large to fit in the memory of an isolated computational resource, and hence, it is split and deployed among the available resources. Every process trains its own portion of the model using the same batch of examples. Depending on the deployment of the model, learners communicate intermediate results using different strategies [13]. As training is an inherently sequential process, this scheme could result in the underexploitation of

the computational resources, as its performance is limited by the communication between the processes and by the number of nodes involved in the training process.

Conversely, data parallelism consists of running *replicas* of the training model on the available computational resources. Every replica holds a local copy of the entire model, which is trained on disjoint data subsets of χ called *batches* [18]. After every batch in the training step is completed, replicas compute the gradients of their respective *loss* function. Next, they coordinate to combine their local computed gradients before updating θ, commonly by tree reduction collective communications [23], or pushing gradients to centralized *parameter servers* [7]. Needless to say, these synchronization points cause straggler processes to have a high impact on the overall performance. Furthermore, performance degradation grows with the heterogeneity of the platform and the number of replicas used to train the model. Model and data types of parallelism can be combined in order to mitigate their limitations in some particular scenarios. This is known as *hybrid parallelism*.

Two general mechanisms contribute to solve the aforementioned performance issue. First, using an asynchronous SGD optimization procedure to relax the consistency of parameter values by allowing processes to asynchronously combine gradients and update parameters. This scheme decouples communication and computation, which highly benefits the training performance. However, the order in which parameters are updated is not deterministic, and hence gradient computations in a replica are done on a stale version of parameter values. This *distance* between a local parameter used to compute gradients in a replica and its global current value is known as *staleness*. Empirical studies [7] show that a low degree of staleness does not penalize the learning accuracy of the model. Meanwhile, other works [10] propose mechanisms for reducing staleness impact on the accuracy of training models.

A second approach to mitigate staleness consists of using load balancing techniques to assign to each replica an amount of work which is in accordance with its computational capacity. A dynamic load balancing mechanism is proposed in [3], in which, in each epoch, every replica is assigned with a batch size proportional to its speed. A key point is to determine the speed of each replica in the system, and this is achieved by using an additional recurrent neural network that calculates the size of the batch in the next epoch as a function of the current speed and processing time. A non-dedicated cloud platform is assumed in this context, with shared assigned computing resources and, hence, variable speed and memory parameters. Although adaptive (and partially able to deal with stragglers due to temporal speed variations), this mechanism requires a large number of epochs to be effective and steals computational resources from the main training process.

In this paper, we introduce a new mechanism for improving the distributed performance of the training process of deep models, while keeping their accuracy. Our methodology follows a two-step approach. First, prior to the training, we use the FuPerMod tool [6] to determine the computational capabilities of the computational resources assigned to each replica in the platform. We assume dedicated heterogeneous clusters made up of both CPUs and GPUs. Secondly, in the training step, we assign a batch size to each replica that is proportional to its relative speed. The goal is to balance the workload and, as a consequence, to minimize the communication of the waiting times of the replicas at the time of communicating the computed

gradients. We use the synchronous SGD with reduction tree communication to combine gradients, which ensures deterministic order in combining the per-replica computed gradients, and hence it does not affect the overall accuracy.

The static load balancing technique has been widely used in the HPC field [1, 21], because it does not require any additional computational resources during the execution time of an application. Notwithstanding this fact, especially in non-dedicated platforms where the workload variation is higher, it can be combined with other mechanisms, such as those proposed in [3] to perform fine-grained and dynamic adaptations during the training stage. Furthermore, this can be combined with asynchronous SGD techniques, such in [4], to ensure minimal staleness. Hence, the primary contributions of this work are:

- A new methodology to improve the performance of data-parallel distributed training of deep models, while preserving the accuracy of the trained model when replicas run on dedicated heterogeneous platforms.
- The application of common HPC-based static workload balancing techniques for training deep models, in order to optimize resource exploitation and execution times.
- An evaluation of the impact of the heterogeneity of dedicated computing platforms on the deep network distributed training process.

The rest of this paper is organized as follows. First, we discuss related works in Sect. 2. Section 3 describes our implementation, including the distribution of processes on the heterogeneous platform and the training model procedure. Section 4 details how we evaluated our system and presents the obtained results. Finally, Sect. 5 presents our conclusions and future work.

2 Related work

The increase in dataset sizes and the number of parameters to learn in deep models have leveraged the usage of HPC platforms to accelerate training, including dedicated clusters and non-dedicated cloud environments. The work in [2] provides an excellent survey of current distributed techniques to parallelize and distribute the training. The main data and model parallelization schemes are described in work [16] that proposes a distribution training of convolutional networks [19] using data parallelism in convolutional layers and model parallelism in fully connected layers, as well as different synchronization methods for parameter updating between workers. In this paper, we focus on the data parallelism scheme applied to convolutional networks.

Paper [7] proposes the *Downpour* asynchronous SGD algorithm implemented in the *DistBelief* framework. This framework enables large-scale model and data parallelism for training purposes. The Downpour algorithm launches multiple replicas of the model using data parallelism, and it uses asynchronous SGD gradient communication based on a centralized parameter server. Authors empirically found that reaching a certain level of staleness tolerance does not significantly impact the model

accuracy. Nevertheless, work [10] proposes a mechanism for reducing the staleness of the asynchronous SGD by modulating the learning rate using the current average gradient staleness values. They provide a discussion on the interplay of hyperparameters and distribution design choices, using the implemented *Rudra* framework. The staleness problem is also addressed in [4], using a different approach. This work performs data-parallel training of models by using *p backup workers* in addition to *P* replicas and data-parallel synchronous optimization. In order to update the model parameters, it considers *P* faster gradient computations and drops the rest. This approach reduces staleness and performance degradation caused by straggler replicas, at the expense of using additional resources. In this paper, we follow a synchronous SGD approximation method that avoids the staleness problem, although it is more sensitive to waiting times at synchronization.

Focusing on heterogeneous platforms, work [15] studies the performance degradation of SGD optimization in heterogeneous platforms (with respect to homogeneous distributed training schemes). They focused on a *Stale Synchronous Parallel* synchronization scheme, in which the grade of staleness is limited by the updating protocol and the parameter server. The authors propose both constant and dynamic learning rate schedules for updating the parameters. In this way, the unstable convergence caused by stragglers is mitigated, improving statistical and hardware performance. A key difference with respect to our work is that we use collective communication as synchronization mechanism, avoiding the necessity of additional parameter server processes.

The thorough work in [3] proposes to adapt batch sizes in each replica to their relative speeds. As a consequence, straggler replicas waiting times are minimized. Authors use a *Bulk Synchronous Parallel* scheme (which avoids staleness) on heterogeneous non-dedicated cloud platforms, with simulated injected stragglers in their experiments. The measurement of the respective speed of the replicas (needed to compute their assigned batch sizes) is achieved using a *Recurrent Neural Network*, trained along epochs with per-replica CPU performance and memory usage values in each iteration. It is worth noting that our work follows a similar approach to those adapting batch sizes in replicas to their computational performance. However, a novelty of our work is to introduce an offline and static load balance mechanism which does not interfere with the training of the model. Work [3] offers additional insights that we plan to include in future works, such as conducting weighted gradient aggregation to avoid per-sample biases in replicas.

3 Detailing the adopted approach

This section details our proposal for training of deep models[1] following a data-parallel scheme. We assume a dedicated heterogeneous platform made up of a set of computational *nodes*, with different speeds. Nevertheless, our approach can also be applied to non-dedicated cloud environments as an initial workload distribution,

[1] The source code is available at https://github.com/mhaut/static_load_deeplearning.

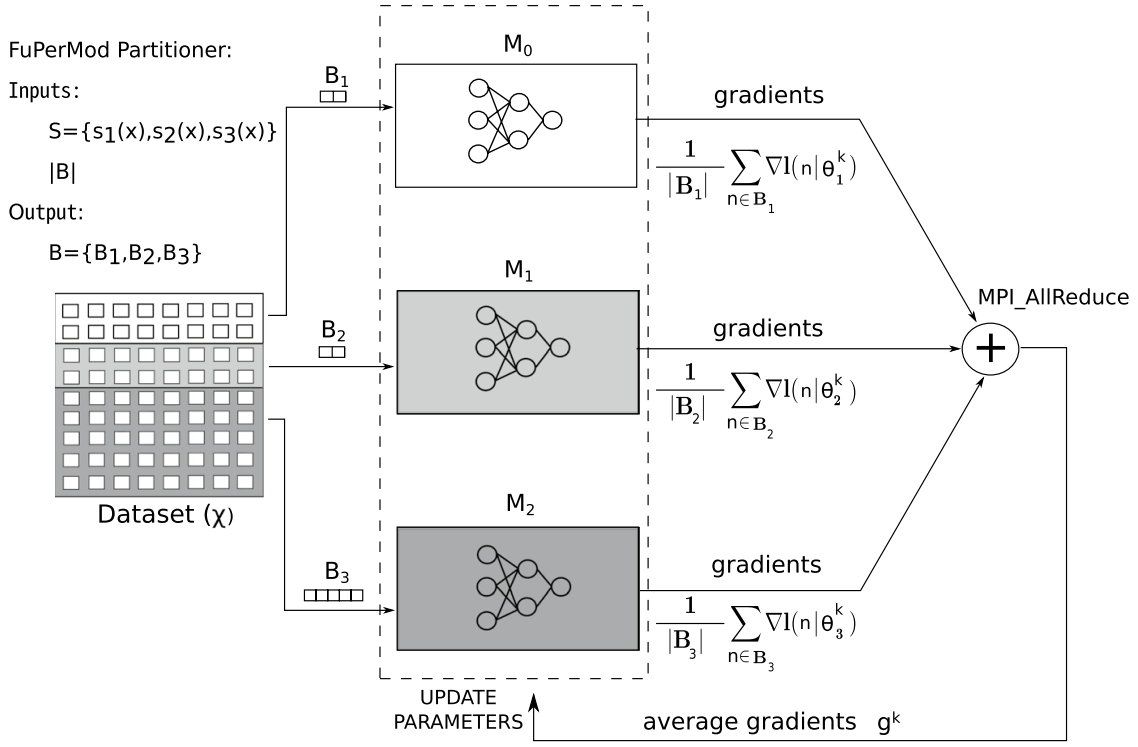


Fig. 1 General distributed learning framework of a model using static load balance mechanisms. The figure represents an iteration k of an epoch in the training process, where $P = 3$ replicas are assigned with an uneven batch size B_p. Replicas perform the parameter update after communicating gradients (g^k) using the `MPI_Allreduce` collective operation

or assuming the possible variations in performance. Replicas running the training process are deployed on the computing nodes of the platform, commonly multicore CPUs and GPUs with different numbers of cores and processing speeds on current heterogeneous platforms.

A key issue is to accurately determine the speed of each node. In this regard, FuPerMod is a tool that is commonly used in heterogeneous HPC platforms. It determines empirically the computational capabilities of a node running a given application. To achieve that, the tool executes a benchmark (provided by the user) in each compute node. This benchmark should be representative of the application in order to obtain meaningful execution profiles. In this work, we use a convolutional neural network (on a range of batch sizes) as the benchmark to measure the speed in each replica. As output, FuPerMod returns the speed of the P computational nodes in the heterogeneous platform, as a set of P functions $S = \{s_1(x), s_2(x), \ldots, s_P(x)\}$. A function $s_p(x)$ varies along a given range of batch sizes x to represent the performance profile of a computational node, which depends on its available resources (including cache and memory sizes). FuPerMod speed is provided as the inverse of the time invested in executing the benchmark for a given batch size $|B|$.

It is important to note that the set of speed functions S characterizing the platform is statically determined in a previous initialization step. Observe Fig. 1. The speed functions, together with the specific batch size $|B|$, are used as inputs to the FuPerMod *partitioner* utility, which computes the number of examples $|B_p|$ assigned to each replica p, with $\sum_{p=1}^{P} |B_p| = |B|$. As FuPerMod *partitioner* works with sizes in

bytes, we convert the input batch size to bytes using $h \times w \times c \times r \times |B|$, with r the number of bytes to represent each pixel.

To train the model, SGD works as a synchronous iterative process over multiple passes of the dataset, called *epochs*. In each epoch, the dataset χ is split up in P subsets and assigned to the P corresponding replicas. The size of the subset in a replica p is computed as $|\chi|/|B| \times |B_p|$. Furthermore, the distribution of the dataset to the replicas is made in such a way that batches assigned to replicas do not overlap in each epoch and, additionally, every replica trains its own copy of the model on the full set of examples along epochs. Then, replicas use their assigned subsets in batches to train their copies of the model. A description of the process is shown in Fig. 1. Iteration k of the SGD training process performs the following general steps:

1. Every replica p manages a batch of $|B_p|$ examples, with $|B| = \sum_{p=1}^{P} |B_p|$. The size of the batch B_p is proportional to the relative speed of the replica p, running on computational resources with speed $s_p(|B_p|)$.
2. Replicas compute their gradients based on a *Loss* function $l(n|\theta_p^k)$ that returns the error of the sample $n \in B_p$, computed in the iteration k using parameters θ_p^k with respect to the actual value. The gradient computation in a replica p is $g_p^k = \frac{1}{|B_p|} \sum_{n \in B_p} \nabla l(n|\theta_p^k)$.
3. After computing the gradient vectors g_p^k, each replica delivers a collective all-reduce communication operation in order to combine gradients as: $g^k = \frac{1}{P} \sum_{p=1}^{P} g_p^k$
4. Each replica updates the parameters using the received gradients and α.

We use PyTorch to train a model based on the previous steps. It implements MPI blocking collectives [8], which impose synchronization, and hence, straggler processes degrade performance. The proposed methodology is also compatible with the use of a *parameter server* that holds an updated global copy of the parameters The main drawback of this approach is its centralized nature that is mitigated by distributing parameters on several server processes [7] and the resources consumed by such additional server processes.

Finally, balancing the workload according to the replicas that determined computational capacity ensures that all replicas finish iteration k at the same time, avoiding idle times at the communication step and hence improving the overall performance. Of course, an error in the workload balancing may always exist, depending on errors in the measurement of the speed of the replicas using benchmarking, and on the granularity of the example size, which we consider negligible. A limitation of this methodology is that a static workload assignment to replicas does not adapt to temporal changes in the system loads derived from the shared usage of resources in non-dedicated platforms. We indeed assume a homogeneous network, and hence, we do not account for the influence of possible imbalances in the gradients communication performance in the model training.

At this point, it is important to note that [3] presents a weighted aggregation in the gradient computation for assigning every example with the same worth, as $g^k = \frac{1}{|B|} \sum_{p=1}^{P} g_p^k$. This improvement does not affect the performance, although it certainly has an impact in the convergence of the model.

4 Experimentation and analysis

This section evaluates the proposed approach. We detail first the hardware and software elements, and then we discuss the results of the training tasks in terms of performance and accuracy. To conclude, we experimentally test the proposed load balanced implementation in ResNet [11], a relevant and standard architecture in the field.

A small test platform called *Metropolis* is used to obtain initial insights and results of the behavior of the implementation and also a dedicated heterogeneous HPC cluster called *Ceta-Ciemat* to obtain real and more complete results of our implementation in a real environment. *Metropolis* is composed of two nodes, *Pluton* and *Caronte*, that use three different GPUs: an NVIDIA RTX 2080Ti and an NVIDIA RTX 2060 (connected to *Pluton*) and an NVIDIA RTX 1050Ti (connected to *Caronte*), with 11 GB, 6 GB and 4 GB of memory, respectively. The CPUs are an Intel Xeon E5620 (Nehalem) 8-core processors running at 2.40 GHz with 12 GB of RAM. Nodes are connected by an Ethernet GigaBit network. Five replicas are deployed in that set of computational nodes. A CPU core is reserved for each replica running on a GPU, in order to be used for memory transfers and communications. On the other hand, the *Ceta-Ciemat* cluster is composed of eight multicore CPU nodes connected by an InfiniBand QDR Network. Each CPU node holds one or two Kepler K80 GPUs with 24 GB of RAM. The CPUs are a 24-core Intel Haswell running at 2.50 GHz with 64 GB of RAM. Although all GPUs are similar, we experimentally found significant differences in speed between nodes using one GPU with respect to nodes using two GPUs. The reason is that GPUs share the PCI bus, which impacts the performance of data transfers between CPUs and GPUs. Such subtle performance penalties can be found in other platforms.

We trained our models using two standard datasets, *MNIST* and *CIFAR-10*. MNIST is composed of black and white handwritten digits images, representing digits from 0 to 9, with a training set of 60.000 examples and a test set of 10.000 examples. These digits have been normalized in size and centered to a fixed image size of $28 \times 28 \times 1$ of 32-bit floats. CIFAR-10 contains 60.000 color images of size $32 \times 32 \times 3$ of 10 non-biased classes. This dataset is used to verify that the proposed implementation provides successful results.

Two models are used in our experiments. Both have a feature extractor composed of several stages of convolutional and pooling layers and a classifier with fully connected (FC) layers. Details are shown in Tables 1 and 2. The two-dimensional convolutional layers have been implemented to extract deep features, employing the rectified linear unit (ReLU) activation function. The feature maps obtained in the convolutional part are reshaped into an unrolled vector representation to feed the classifier.

As a conduit example, we evaluate the performance of training the model described in Table 1 with the MNIST dataset in the *Metropolis* platform. Performance is measured using wall clock time per epoch. The final results are obtained by taking the maximum training time per epoch of the replicas involved

Table 1 Layers of the convolutional neural network implemented for classification of the MNIST image dataset

Model for MINST (Number of parameters: 21,840)				
Layer ID	Kernel/neurons	Activation funct.	Pooling	Dropout
Conv1	$5 \times 5 \times 10$	ReLU	2×2	No
Conv2	$5 \times 5 \times 20$	ReLU	2×2	No
FC1	50	ReLU	–	Yes
FC2	n_{classes}	Softmax	–	No

Padding is added to convolutional layers for not shrinking the image

Table 2 Layers of the convolutional neural network for the classification of the CIFAR-10 image dataset

Model for CIFAR-10 (Number of parameters: 176,034)				
Layer ID	Kernel/neurons	Activation funct.	Pooling	Dropout
Conv1	$7 \times 7 \times 10$	ReLU	2×2	No
Conv2	$7 \times 7 \times 20$	ReLU	2×2	No
FC1	120	ReLU	–	No
FC2	84	ReLU	–	No
FC3	n_{classes}	Softmax	–	No

Padding is added to convolutional layers to avoid shrinking the images

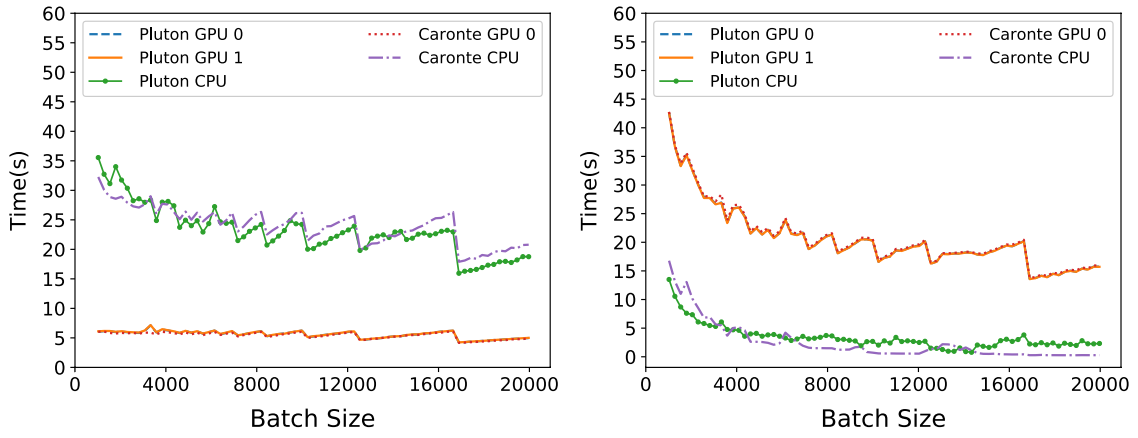


Fig. 2 Computation (left) and communication (right) times of training the model described in Table 1 on the MNIST dataset in *Metropolis* platform for a range of batch sizes, evenly distributed between $P = 5$ replicas. The processing time is provided in seconds per epoch

in the training process, along five different executions. To obtain our performance data, we run $e = 10$ epochs. As a baseline test, we use a homogeneous (unbalanced) distribution of the batch size $|B|$ between replicas in the platform, with $|B_p| = |B|/P, \forall p$. Figure 2 shows the performance data in separate computation and communication plots, for a range of batch sizes $|B|$. The communication time is measured from the invocation of the blocking communication operation to the reception of the gradients, and it includes waiting times. As expected, replicas

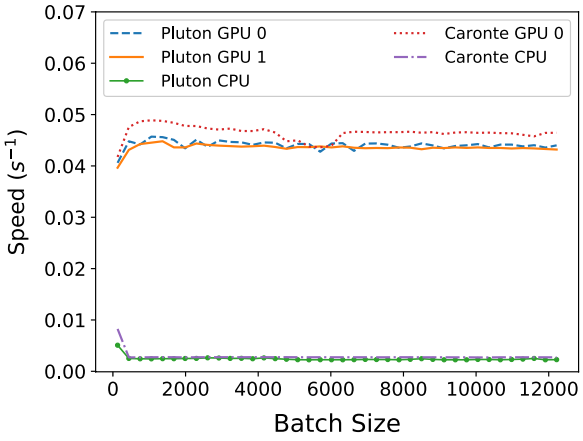


Fig. 3 FuPerMod characterization of the computational capabilities of $P = 5$ replicas running on the *Metropolis* nodes. Speeds are shown for a range of batch sizes $|B|$

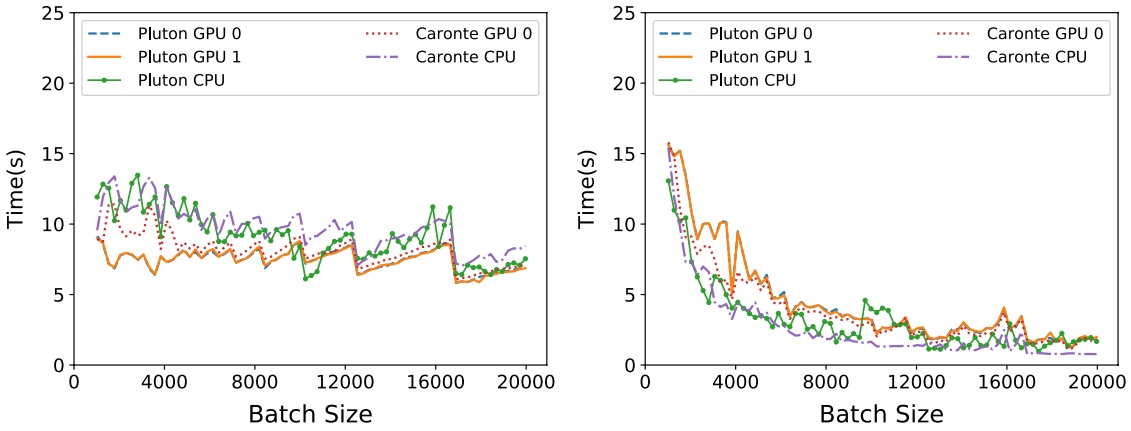


Fig. 4 Computation (left) and communication (right) times of training the model described in Table 1 on the MNIST dataset in *Metropolis* platform for a range of batch sizes. Batch sizes are unevenly distributed between $P = 5$ replicas, according to their speeds determined by FuPerMod tool. Time is provided in seconds per epoch

running on CPUs spend more time in computing their batches of examples than those on GPUs, because of their lower computing throughput. As a consequence, GPU replicas wait at communication points, degrading overall performance of the training process.

Figure 3 illustrates the speeds of the replicas running on the *Metropolis* computational nodes, obtained by benchmarking using FuPerMod. As expected, high differences between GPUs and CPUs are observed, with slight differences between processes running in similar resources (either CPUs or GPUs). Figure 4 shows the results of the training execution for a range of batch sizes $|B|$, under the proposed static balanced distribution. FuPerMod *partitioner* unevenly distributed every batch B between replicas according to their speeds. Due to the fact that every replica needs to complete a similar amount of computing work, communication waiting times are dramatically reduced with respect to Fig. 2. This reduction also arises from the reduction in the differences of the time lines in the plots. As a consequence, the overall performance is significantly improved, as shown in Fig. 5.

We now focus on the *Ceta-Ciemat* system to validate performance results obtained in the previous proof-of-concept *Metropolis* platform. Figure 6 (top)

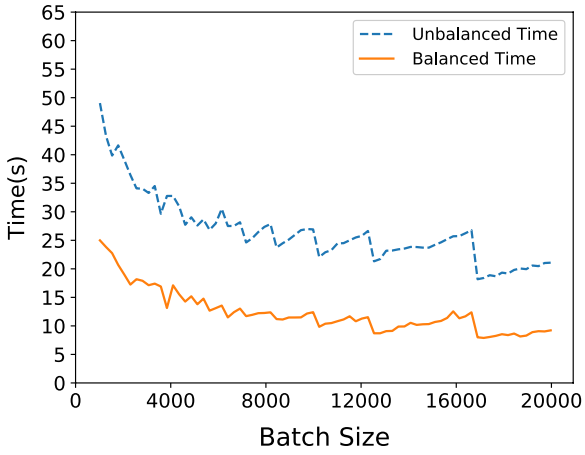


Fig. 5 Performance of training the model described in Table 1 on the MNIST dataset for unbalanced and balanced distributions between $P = 5$ replicas in *Metropolis*

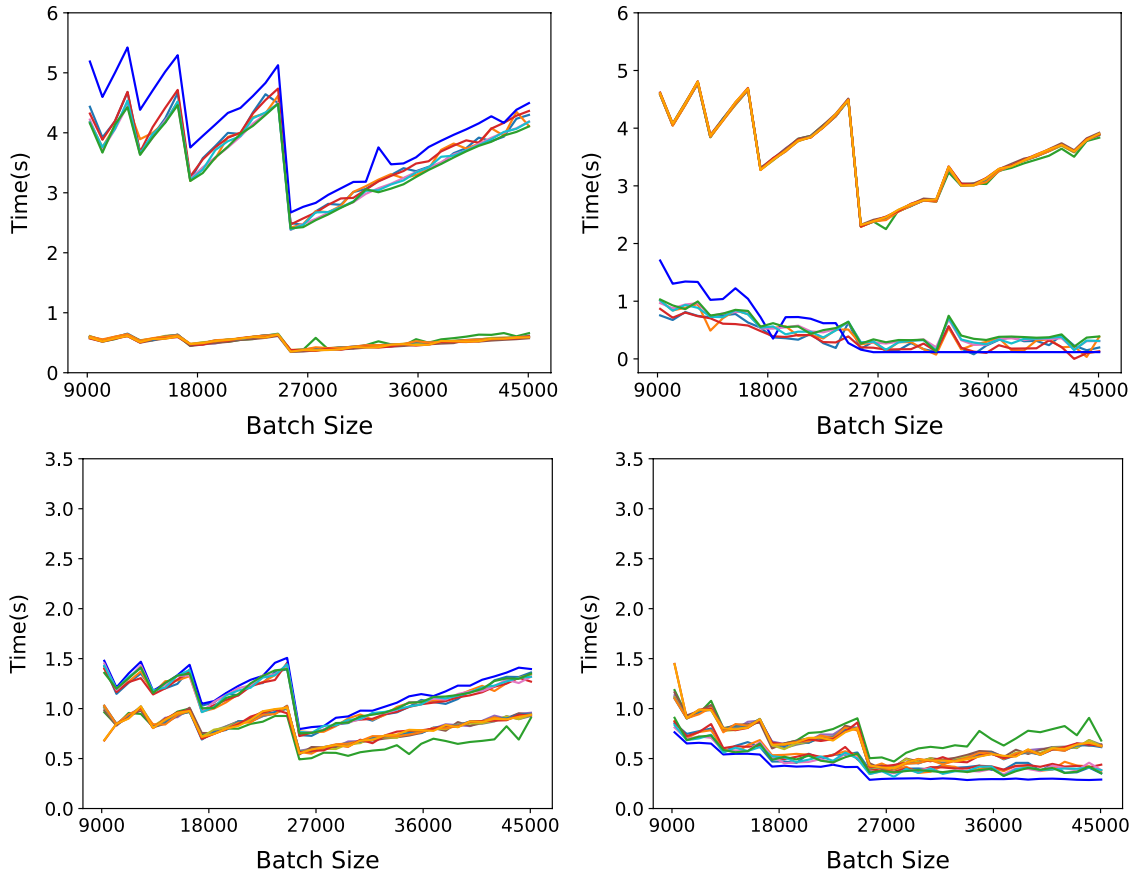


Fig. 6 Performance times in seconds per epoch of the computation (left) and communication (right) of training the model in Table 2, using CIFAR-10 dataset in the *Ceta-Ciemat* cluster, for increasing batch sizes. Top figures represent evenly (unbalanced) distribution and bottom figures represent unevenly (balanced) distribution between $P = 16$ replicas

shows the performance results obtained after training the model detailed in Table 2 with the CIFAR-10 dataset. We use a homogeneous distribution of the batch size, hence with $|B_p| = |B|/P, \forall p$, between $P = 16$ replicas deployed on available computational nodes of the platform. As in the previous platform, we have two groups of replicas depending on whether they run on CPUs or GPUs. Differences in performance between groups are high; however, slight differences

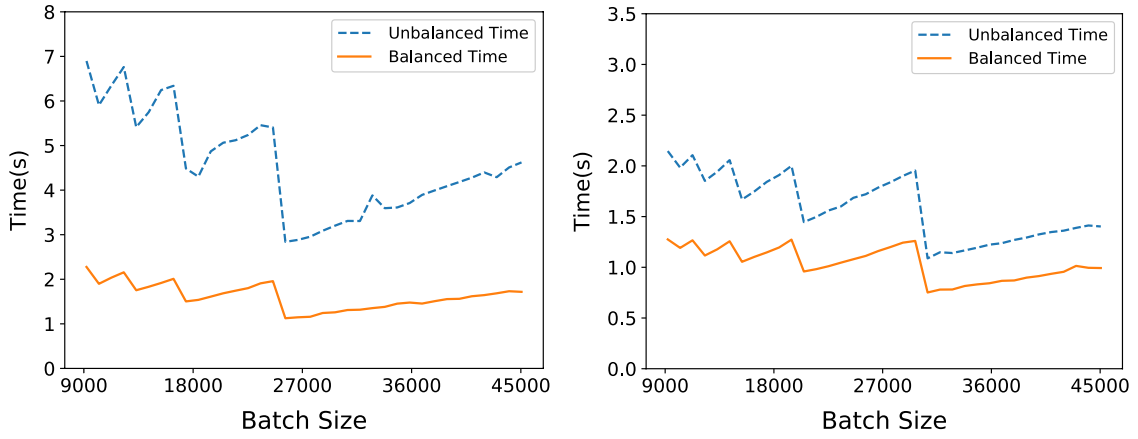


Fig. 7 Performance times in seconds per epoch for models trained with CIFAR-10 (left) and MNIST (right) datasets on the *Ceta-Ciemat* platform ($P = 16$ replicas) for unbalanced and balanced distributions of batch sizes between $P = 16$ replicas

in performance between replicas running on similar resources also impact the performance of the entire training process. Note that differences in computation (and hence, inversely, in communication, due to waiting times) grow with the batch sizes. The corresponding results obtained with a balanced distribution of the batch $|B|$ between replicas, according to their speeds, are shown at the bottom of Fig. 6. Again, plot lines differences shrink, meaning that replicas invest similar times in performing their assigned computing workloads, hence reducing waiting times at communication points. Figure 7 displays the total performance times for the two models in Tables 1 and 2, trained, respectively, on CIFAR-10 and MNIST on the *Ceta-Ciemat* platform with $P = 16$ replicas. Differences between unbalanced and balanced distributions remain constant along batch sizes for both MNIST and CIFAR-10 datasets. As a summary, the average speedups along the range of batch sizes for statically balanced workload distributions—with respect to unbalanced ones—are 1.52% and 2.78%, respectively.

Additionally, we evaluate the *accuracy*, defined as the percentage of correct classification of examples, given non-biased datasets as MNIST and CIFAR-10. In order to show the behavior of the models in both unbalanced (even) and balanced (uneven) workload distributions, the batch size is set to $|B| = 2048$ examples. Figure 8 shows both per-epoch and temporal evolution of the accuracy of the model trained on MNIST. We include the temporal evolution of accuracy to illustrate differences in convergence times of the methods with respect to training time. We can observe that both distribution approaches require around $e = 50$ epochs to converge to a similar accuracy. Figure 9 shows the corresponding results for CIFAR-10, with the difference that the model needs $e = 300$ epochs to converge on this (larger) dataset. We actually executed a higher number of epochs than those shown in both models plots; however, we experimentally observe that those values of e were enough for the models to converge. Table 3 summarizes the accuracy of the models including their standard deviation along five executions. The table shows the accuracy reached for MNIST and CIFAR-10 datasets in the *Ceta-Ciemat* platform for $e = 50$ and $e = 300$ epochs, respectively. Accuracy figures fall in the same range if we consider

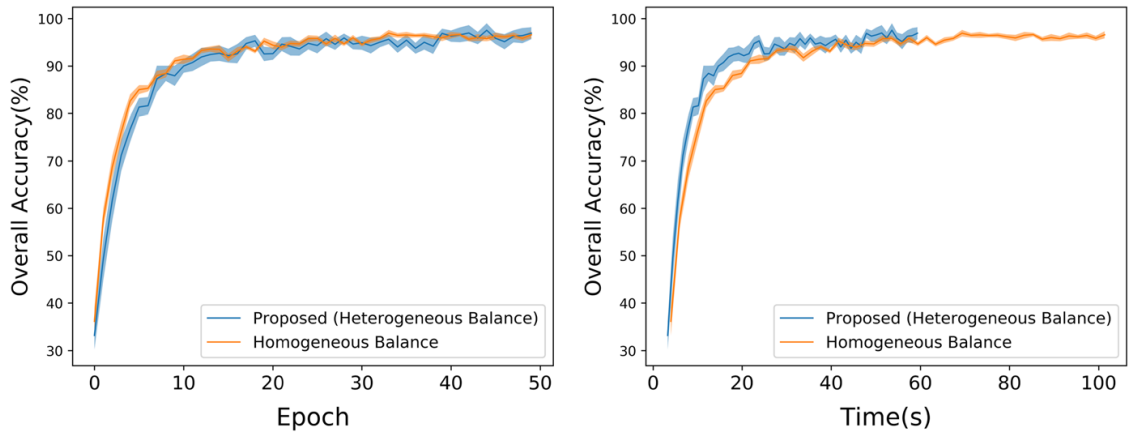


Fig. 8 Accuracy of the model described in Table 1, trained on MNIST dataset for unbalanced and balanced distributions of batch sizes $|B|$ in the *Ceta-Ciemat* platform. Both per-epoch and temporal evolution of the accuracy are shown. The shaded areas represent the standard deviation along a set of five repetitions

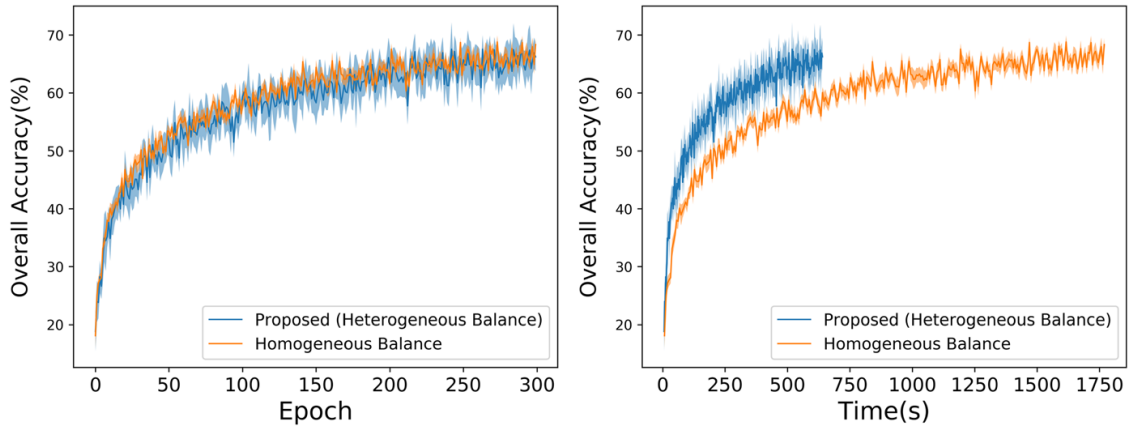


Fig. 9 Accuracy of the model described in Table 2 trained on CIFAR-10 dataset for unbalanced and balanced distribution of batch sizes $|B|$ in the *Ceta-Ciemat* platform. Both per-epoch and temporal evolution of the accuracy are shown. The shaded areas represent the standard deviation along a set of five repetitions

Table 3 *Maximum* accuracy and *Last* accuracy (obtained in the last epoch) of models described in Table 1 and Table 2, trained on MNIST and CIFAR-10 datasets, respectively

Dataset	Maximum accuracy		Last accuracy		#Epochs
	Unbalanced	Balanced	Unbalanced	Balanced	
MNIST	**97.54** ± 1.47	96.92 ± 0.69	**96.92** ± 1.26	96.58 ± 0.75	$e = 50$
CIFAR-10	67.54 ± 4.64	**68.80** ± 1.11	66.22 ± 2.45	**68.34** ± 1.10	$e = 300$

The best metric values are in bold

the standard deviation; hence, there are no significant differences. Nevertheless, the aggregation of weighted gradients in the trained process (proposed in work [3] and described in Sect. 3) should make accuracy values even closer, without affecting

Table 4 Training times for several ResNet architectures

Network	Parameters (M)	Unbalanced	Balanced	Speedup
ResNet20	0.27	65.31	**25.67**	2.54
ResNet32	0.46	103.84	**41.76**	2.49
ResNet44	0.66	145.10	**58.25**	2.49
ResNet56	0.85	185.64	**74.24**	2.50
ResNet110	1.72	853.50	**104.07**	8.20

The best metric values are in bold

Parameters column shows the number of trainable parameters (in millions). Times for unbalance and balance workload distributions are in seconds per epoch. Last column shows obtained speedups

performance. Finally, as the previous results show that the accuracy remains constant with load balancing, performance tests have been carried out on the main ResNet architectures under CIFAR-10. Table 4 shows that the speedup obtained remains constant when the number of parameters is not triggered, while with a higher number of parameters it achieves a higher speedup.

5 Conclusions and future work

This paper describes a new approach to distribute the training of deep networks on dedicated heterogeneous platforms. The proposed methodology departs from a precomputed characterization of the speed of the computational resources of the platform. It provides replicas of the model (running in such computational resources) with a batch size that is proportional to their computing capabilities, in such a way that waiting times at communication points (performed at each training iteration to combine gradients between processes) are eliminated. As a consequence, the overall execution time needed for training the deep model is reduced (while the accuracy is not significantly affected), as demonstrated for two different platforms and datasets.

One of the main contributions of this work is the exploitation of well-known HPC optimization techniques to balance the distributed training of deep learning models. Previous works propose dynamic load balancing techniques that impact the performance, albeit requiring a high number of iterations to be really effective. In turn, our methodology can dynamically adapt batch sizes to performance variations during the training process, as a result from the non-dedicated usage of resources (which is characteristic of cloud environments).

Our future work will focus on the study of the scalability of the proposed implementation regarding the number of replicas, using both larger datasets and deeper networks.

Acknowledgements This work was jointly supported by the following projects and institutions: (1) The European Regional Development Fund 'A way to achieve Europe' (ERDF) and the Extremadura Local Government (Ref. IB16118). (2) The Ministry of Education, November 19, 2015, of the Secretary of State for Education, Vocational Training and Universities, under grant FPU15/02090. (3) The computing facilities

of Extremadura Research Center for Advanced Technologies (CETA-CIEMAT), funded by the European Regional Development Fund (ERDF).

References

1. Beaumont O, Boudet V, Rastello F, Robert Y (2001) Matrix multiplication on heterogeneous platforms. IEEE Trans Parallel Distrib Syst 12(10):1033–1051. https://doi.org/10.1109/71.963416
2. Ben-Nun T, Hoefler T (2018) Demystifying parallel and distributed deep learning: an in-depth concurrency analysis. arXiv:1802.09941
3. Chen C, Weng Q, Wang W, Li B, Li B (2018) Fast distributed deep learning via worker-adaptive batch sizing. In: Proceedings of the ACM Symposium on Cloud Computing, SoCC '18. ACM, New York, USA, pp 521–521
4. Chen J, Monga R, Bengio S, Jozefowicz R (2016) Revisiting distributed synchronous SGD. In: ICLR Workshop Track
5. Chiu C, Sainath TN, Wu Y, Prabhavalkar R, Nguyen P, Chen Z, Kannan A, Weiss RJ, Rao K, Gonina K, Jaitly N, Li B, Chorowski J, Bacchiani M (2017) State-of-the-art speech recognition with sequence-to-sequence models. arXiv:1712.01769
6. Clarke D, Zhong Z, Rychkov V, Lastovetsky A (2013) Fupermod: a framework for optimal data partitioning for parallel scientific applications on dedicated heterogeneous HPC platforms. In: Parallel Computing Technologies. Springer, Berlin, Heidelberg, pp 182–196
7. Dean J, Corrado GS, Monga R, Chen K, Devin M, Le QV, Mao MZ, Ranzato M, Senior A, Tucker P, Yang K, Ng AY (2012) Large scale distributed deep networks. In: NIPS, USA, pp 1223–1231
8. Forum MPI (2015) MPI: a message-passing interface standard, version 3.1 , June 4, 2015. High-Performance Computing Center Stuttgart, University of Stuttgart
9. Fox G, Qiu J, Jha S, Ekanayake S, Kamburugamuve S (2016) Big data, simulations and HPC convergence. In: Big Data Benchmarking. Springer, Cham, pp 3–17
10. Gupta S, Zhang W, Wang F (2017) Model accuracy and runtime tradeoff in distributed deep learning: a systematic study. In: IJCAI, pp 4854–4858
11. He K, Zhang X, Ren S, Sun J (2015) Deep residual learning for image recognition. arXiv:1512.03385
12. Hornik K (1991) Approximation capabilities of multilayer feedforward networks. Neural Netw 4(2):251–257
13. Huang Y, Cheng Y, Chen D, Lee H, Ngiam J, Le QV, Chen Z (2018) Gpipe: efficient training of giant neural networks using pipeline parallelism. arXiv:1811.06965
14. Jain AK, Mao J, Mohiuddin KM (1996) Artificial neural networks: a tutorial. Computer 29(3):31–44
15. Jiang J, Cui B, Zhang C, Yu L (2017) Heterogeneity-aware distributed parameter servers. In: Proceedings of the 2017 ACM International Conference on Management of Data, SIGMOD '17. ACM, NY, USA, pp 463–478
16. Krizhevsky A (2014) One weird trick for parallelizing convolutional neural networks. arXiv:1404.5997
17. Krizhevsky A, Sutskever I, Hinton GE (2012) Imagenet classification with deep convolutional neural networks. In: Advances in Neural Information Processing Systems 25. Curran Associates, Inc., pp 1097–1105
18. Le QV, Ngiam J, Coates A, Lahiri A, Prochnow B, Ng AY (2011) On optimization methods for deep learning. In: Proceedings of the 28th International Conference on International Conference on Machine Learning, ICML'11. Omnipress, USA, pp 265–272
19. LeCun Y, Bengio Y, Hinton G (2015) Deep learning. Nature 521:436
20. Paoletti M, Haut J, Plaza J, Plaza A (2019) Deep learning classifiers for hyperspectral imaging: a review. ISPRS J Photogramm Remote Sens 158:279–317
21. Rico-Gallego JA, Díaz-Martín JC, Calvo-Jurado C, Moreno-Álvarez S, García-Zapata JL (2019) Analytical communication performance models as a metric in the partitioning of data-parallel kernels on heterogeneous platforms. J Supercomput 75(3):1654–1669
22. Schmidhuber J (2015) Deep learning in neural networks: an overview. Neural Netw 61:85–117
23. Sergeev A, Balso MD (2018) Horovod: fast and easy distributed deep learning in TensorFlow. arXiv:1802.05799

Affiliations

Sergio Moreno-Álvarez[1] · Juan M. Haut[2] · Mercedes E. Paoletti[2] · Juan A. Rico-Gallego[1] · Juan C. Díaz-Martín[2] · Javier Plaza[2]

Juan M. Haut
juanmariohaut@unex.es

Mercedes E. Paoletti
mpaoletti@unex.es

Juan A. Rico-Gallego
jarico@unex.es

Juan C. Díaz-Martín
juancarl@unex.es

Javier Plaza
jplaza@unex.es

[1] Department of Computer Systems Engineering and Telematics, University of Extremadura, Cáceres, Spain

[2] Department of Technology of Computers and Communications, University of Extremadura, Cáceres, Spain

Escuela Politecnica
Av. de la Universidad, S/N, 10003
Caceres, Spain
Phone: 0034927257000. Ext. 51655
Email: jarico,juanmariohaut{@unex.es}

Dr. Juan Antonio Rico Gallego y Dr. Juan Mario Haut Hurtado como directores de la tesis titulada "Desarrollo de técnicas de aprendizaje automático para la optimización de aplicaciones científicas y de procesamiento masivo de datos en entornos de altas prestaciones", acreditan la colaboración y aportación del doctorando en el trabajo. Asímismo, se certifica también el factor de impacto junto con la respectiva categoría de la siguiente publicación incorporada en la tesis doctoral. Para cualquier aclaración con respecto a lo indicado, por favor, contacten con cualquiera de los directores.

Juan Antonio Rico Gallego PhD and Juan Mario Haut Hurtado PhD as directors of the PhD thesis titled "Development of machine learning techniques for the optimization of scientific and massive data processing applications in high performance computing environments", certify the collaboration and contribution of the doctoral student at work. Likewise, the impact factor is also certified along with the respective category of the following publication incorporated in the doctoral thesis. For any clarification regarding what is indicated, please contact any of the directors.

Artículo / Paper

Authors: **S. Moreno-Álvarez**, J. M. Haut, M. E. Paoletti and J. Rico-Gallego.
Title: Heterogeneous model parallelism for deep neural networks.
Journal: Neurocomputing.
Other Information: vol 441, pp 1-12, 2021.
DOI: 10.1016/j.neucom.2021.01.125.
Impact factor 2020: 5.719. Q1
Abstract: Deep neural networks (DNNs) have transformed computer vision, establishing themselves as the current state-of-the-art for image processing. Nevertheless, the training of current large DNN models is one of the main challenges to be solved. In this sense, data-parallelism has been the most widespread distributed training strategy since it is easy to program and can be applied to almost all cases. However, this solution suffers from several limitations, such as its high communication requirements and the memory constraints when training very large models. To overcome these limitations model-parallelism has been proposed, solving the most substantial problems of the former strategy. However, describing and implementing the parallelization of the training of a DNN model across a set of processes deployed on several devices is a challenging task. Current proposed solutions assume a homogeneous distribution, being impractical when working with devices of different computational capabilities, which is quite common on high performance computing platforms. To address previous shortcomings, this work proposes a novel model-parallelism technique considering heterogeneous platforms, where a load balancing mechanism between uneven devices of an HPC platform has been implemented. Our proposal takes advantage of the Google Brain's Mesh-TensorFlow for convolutional networks, splitting computing tensors across filter dimension in order to balance the computational load of the available devices. Conducted experiments show an improvement in the exploitation of heterogeneous computational resources, enhancing the training performance. The code is available on: `https://github.com/mhaut/HeterogeneusModelDNN`.

Contribución del doctorado: Planteamiento de la hipótesis, desarrollo práctico, análisis y discusión de los resultados, elaboración y escritura del manuscrito.

Firma / Signature
Nov / Nov, 2021

Juan Antonio Rico Gallego

Juan Mario Haut Hurtado

Neurocomputing 441 (2021) 1–12

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

Heterogeneous model parallelism for deep neural networks

Sergio Moreno-Alvarez [a,*], Juan M. Haut [b], Mercedes E. Paoletti [c], Juan A. Rico-Gallego [a]

[a] Media Engineering Group (GIM), Department of Computer Systems Engineering and Telematics, Escuela Politécnica, University of Extremadura, 10003 Cáceres, Spain
[b] Department of Communication and Control Systems, Higher School of Computer Engineering, National Distance Education University, Madrid, Spain
[c] Hyperspectral Computing Laboratory, Department of Technology of Computers and Communications, Escuela Politécnica, University of Extremadura, 10003 Cáceres, Spain

ARTICLE INFO

Article history:
Received 13 August 2020
Revised 23 January 2021
Accepted 29 January 2021
Available online 23 February 2021
Communicated by Zidong Wang

Keywords:
Deep learning
High performance computing
Distributed training
Heterogeneous platforms
Model parallelism

ABSTRACT

Deep neural networks (DNNs) have transformed computer vision, establishing themselves as the current state-of-the-art for image processing. Nevertheless, the training of current large DNN models is one of the main challenges to be solved. In this sense, *data-parallelism* has been the most widespread distributed training strategy since it is easy to program and can be applied to almost all cases. However, this solution suffers from several limitations, such as its high communication requirements and the memory constraints when training very large models. To overcome these limitations *model-parallelism* has been proposed, solving the most substantial problems of the former strategy. However, describing and implementing the parallelization of the training of a DNN model across a set of processes deployed on several devices is a challenging task. Current proposed solutions assume a homogeneous distribution, being impractical when working with devices of different computational capabilities, which is quite common on high performance computing platforms. To address previous shortcomings, this work proposes a novel model-parallelism technique considering heterogeneous platforms, where a load balancing mechanism between uneven devices of an HPC platform has been implemented. Our proposal takes advantage of the Google Brain's Mesh-TensorFlow for convolutional networks, splitting computing tensors across filter dimension in order to balance the computational load of the available devices. Conducted experiments show an improvement in the exploitation of heterogeneous computational resources, enhancing the training performance. The code is available on: https://github.com/mhaut/HeterogeneusModelDNN.

1. Introduction

Nowadays, those techniques based on neurobiology structures and deep learning (DL) [1] approaches have completely transformed the methods of automatic data analysis within different fields such as robotics, medicine, industry, remote sensing, intelligence and defense, among others [2–5]. Shallow and deep artificial neural networks (ANNs and DNNs, respectively) are powerful feature extraction models due to their ability to automatically learn the hidden patterns and internal regularities/relationships from large collections of examples. These structures are composed of interconnected computational nodes (known as neurons), where each one serves as a linear discriminant. Furthermore, these neurons are arranged in hierarchically stacked layers. Input data is driven through each layer, which apply a trainable set of weights to detect and respond to certain stimuli (i.e., input features, such as spectral attributes as light intensity or spatial patterns as borders and edges). Indeed, those weights are automatically learned to fit the data.

In particular, *Convolutional Neural Networks* (CNNs) [6] efficiently process multidimensional input arrays by processing local receptive fields on the data. Within each convolutional layer, the weights are organized as multidimensional filters (also called kernels) which are slid over the input data volume following a stride through its different dimensions. As a result of convolving the filters over the input, an output feature volume is obtained, identifying the presence of the filtered feature and locating its position.

CNNs achieve competitive performances, establishing themselves as the current state-of-the-art in image processing and computer vision techniques [7,8]. However, current implementations have dramatically grown in both depth and structural complexity. This imposes severe restrictions on the hardware resources used to train the models, due to their high computational burden and massive memory consumption [9]. As a consequence, training a CNN model is a compute-intensive task that often takes several days even using specialized hardware as GPUs. Traditional techniques, such as data batching [10], have provided an adequate solution to the problem by dividing training data into smaller pieces that

* Corresponding author.

https://doi.org/10.1016/j.neucom.2021.01.125
0925-2312/

can be easily handled by the computer system. However, as the amount of data and the structural complexity of networks increases, new techniques are needed.

One of the most widely used methods to accelerate the training stage is *data parallelism*. Data parallelism consists of running several *replicas* of the training model on every device in a parallel system. Every replica holds a local copy of the entire model, which are trained on disjoint data batches assigned to each replica. In every batch step, replicas communicate computed gradients in order to update local weights. Communication between replicas is achieved using two main methods: collective communications [11], or centralized *parameter servers* [12]. Some works [13–15] have achieved a very high scaling efficiency using data parallelism approaches, indicating that communication overhead is not excessive. Nonetheless, some data parallelism applications may suffer excessive inter-device communication overhead, which severely hinders the scalability of training large models [16,17].

An alternative is *model parallelism*. This scheme is used when the model is too large to fit in the memory of a single device, and hence data parallelism cannot be used. The model to train is partitioned and every device trains its own portion of the model using the same batch of examples. Depending on the model deployment, devices communicate intermediate results using different strategies [18]. In this parallelization scheme, communication is usually less than in the data parallelism scheme.

Nevertheless, in both data and model parallelism schemes, synchronization points imposed by the communication cause straggler processes to have a high impact on the overall performance, which is a well-known problem in parallel HPC applications. Furthermore, performance degradation grows with the heterogeneity of the platform and the number of devices used to train the model, which limit the scalability of these approaches.

To mitigate performance differences between devices, and hence diminishing straggler processes impact, *load balancing* techniques are used. These techniques assign a quantity of work to each device according to their computational capabilities. These techniques have been applied to data parallelism training of large models. A dynamic load balancing using an adaptive batch size proportional to every device speed is proposed in [19] and evaluated on a non-dedicated platform, with shared computing resources and variable speed and memory. On tightly-coupled platforms, a static approach is proposed in [20] using a neural network to determine the batch size proportional to the speed of every resource.

In this paper, a static load balancing approach is proposed for the model parallelism scheme, with the goal of the efficient training of deep neural architectures on heterogeneous HPC platforms. Our proposal is based on Mesh-TensorFlow [21]. This framework assumes a homogeneous platform and proposes an uniform distribution of the workload among all the devices involved in the training. Our main contribution is the extension of the previous framework to heterogeneous platforms, such as those composed of CPUs and GPUs of different capabilities and features, currently widespread in HPC. Our approach seeks to minimize the waiting times of the fastest devices at the communication points during training.

The methodology consists in a pre-training phase implementation for static heterogeneous partitioning of the convolutional filters, assigning to each device a number of filters determined by its computational capabilities. This is done using an HPC technique to measure the computational capabilities of every device. This approach is automatic and transparent to the user. Another contribution is the possibility of a heterogeneous model partitioning across the number of filters i.e. the channel dimension. Logically, the contributions maintain the capabilities of Mesh-TensorFlow to homogeneously partitioning the height and width dimensions of a tensor. This scheme of heterogeneous model parallelism improves the performance on heterogeneous HPC platforms by adequately exploiting resource capabilities, while it does not affect negatively to the overall accuracy.

The paper is structured as follow. Section 2 details related state of the art articles. Section 3 introduces the Mesh-TensorFlow methodology. Section 4 explains the heterogeneous approach and the followed training procedure. Section 5 evaluates the performance of the proposed methodology. In this sense, the impact of filter partitioning in CNNs is evaluated on two platforms composed of many heterogeneous devices to demonstrate its behaviour on different environments. MNIST and CIFAR100 have been used to test the performance. Testing on MNIST has been done using a three-layers CNN, while for CIFAR100 different versions of the VGG network [22] has been implemented. Promising results demonstrates that the proposed methodology reduces training times while maintains accuracy by avoiding waiting times at communication points. Finally, Section 6 shows our conclusions and possible extensions.

2. Related work

Training large scale models is a compute intensive task usually performed in HPC platforms with multiple computational nodes to accelerate the process. Recent advantages in speeding up CNNs training are shown in [23]. This work focuses on studying different forms of acceleration when hardware is limited and an improvement in performance is sought. A thorough study of the parallelization training schemes and issues is carried out in Tal Ben-Nun et al. [24].

Parallelization schemes are commonly classified in *data parallelism* and *model parallelism* [25,12]. While our work focuses on the model parallelism scheme on heterogeneous platforms composed of both CPU and GPU nodes, several works addressed data parallelism and provide with common methods and techniques to parallelize and speedup training of large models. Jiang et al. [26] study the performance degradation of SGD optimization in heterogeneous platforms with respect to homogeneous distributed training schemes. They proposed the *Stale Synchronous Parallel* synchronization scheme to address the problem known as *staleness*, a consistency issue derived from the different speeds of parameter updating by processes. The grade of staleness is limited by the weights updating protocol and the parameter servers. An asynchronous distributed version with variance reduction is proposed in [27]. Load balancing is an alternative method to address staleness issue. Chen et al. [19] proposes to adapt batch sizes in each replica training the model in a device to their relative speeds. As a consequence, straggler replicas waiting times are minimized. Authors use a *Bulk Synchronous Parallel* training scheme on heterogeneous non-dedicated cloud platforms with simulated injected stragglers in their experiments. The measurement of the respective speed of the replicas needed to compute their assigned batch sizes is achieved using a *Recurrent Neural Network*, trained along epochs with a per-replica CPU performance and memory usage values in each iteration. Moreno et al. [28] proposes a static load balancing technique in heterogeneous HPC clusters dedicated platforms, assigning to each replica a batch size proportional to its relative speed. The speeds of the replicas are determined prior to the training step by using a benchmark and the FuPerMod tool [29].

Regarding model parallelism, this approach has been traditionally used for big models that do not fit in the memory of a single device [30]. An efficient intra-layer model parallel approach that enables training models with billions of parameters is implemented in [31]. However, partitioning the model is still a complex task and usually it requires to optimize memory for a very high

number of parameters [32]. A foundational work with a novel approach for automatically placing operations of a large model on the devices of a parallel heterogeneous system is based on the *reinforcement learning* paradigm [33].

An approach to obtaining speedup is to split up a model across multiple devices using pipelining. *Pipedream* [34] proposes to adopt pipelining by concurrently injecting multiple mini-batches to the model. It smooths out the consistency problem ensuring that each device over a mini-batch sees the same weights by storing multiple versions of weights. *Gpipe* framework [18] proposes a pipelining method by dividing each mini-batch into multiple micro-batches and indeed pipelining the micro-batches of the same mini-batch to reduce the staleness problem. DNNs introduce significant efforts for programmers to partition model layers. Harlap et al. [34] determine the profiles of the processing time of each layer offline and use dynamic programming to create balanced partitions of the model though a pipeline.

The literature includes multiple frameworks which are responsible for providing data and model based parallelization schemes through different programming languages [35]. *Horovod* [36] proposed distributed training of models using data parallelism in convolutional layers and model parallelism in fully connected layers with different synchronization methods for the parameter updating phase. Users should indicate which device is responsible for executing each of the different layers of the model. *DistBelief* [37] framework is used for deep models in a cluster of hundreds of nodes. It provides model parallelism for one or multiple machines. Furthermore, it provides data parallelism under two different distributed methods. *DownpourSGD* method is an asynchronous stochastic gradient descent procedure which leverages adaptive learning rates used for multiple replicas, and *Sandblaster L-BFGS* method is a distributed implementation of L-BFGS using a type of hybrid parallelization (joint data and model parallelism). *PyTorch* [38] framework provides an efficient set of primitives as well as a library for model parallelism based on remote procedure calls. Another parallelization framework called EC-DNN is proposed in [39] to tackle the degradation of the global model that the average parameters of the local models produce. In this sense, Sun and Liu [39] introduces the weighted sum instead of the average of the outputs of the local models in order to ensure that the global model perform better than local models.

Recently, new workload distribution methods have been implemented to address the requirements of training deep networks. Guang Shi [40] propose a model parallelism approach to efficiently train Deep Belief Networks (DBNs) using a high performance distributed environment. This proposal have managed to accelerate significantly the training over different datasets. Mesh-TensorFlow [21] specifies a language to distribute the workload among the different available devices. It provides both data parallelism and model parallelism schemes. Under the model parallelism scheme, and assuming a homogeneous platform, it evenly distributes tensor computations across a specified dimension on the available devices.

Our work is based on Mesh-TensorFlow. It focuses on model parallelism scheme for convolutional layers, performing heterogeneous partitions and distributions of tensors based on the capabilities of each device in a dedicated HPC platform. This static load-balancing technique reduces the waiting times of processes at synchronization points, and as a consequence, increases the training performance in heterogeneous platforms. Thus, the homogeneous proposal of Mesh-TensorFlow [21] is our comparison objective in terms of performance.

3. Mesh-TensorFlow methodology

Mesh-TensorFlow is a language for specifying distributed tensor computations between a set of processes deployed on the homogeneous computational resources of an HPC platform. Mesh-TensorFlow is built on top of TensorFlow [41], and hence inherits most of its constructs, as graphs, tensors, variables, devices and the automatic computation of gradients. Furthermore, it extends TensorFlow with three main components: tensors, meshes and layouts. Each one is described in detail below.

A *tensor* in Mesh-TensorFlow is a multidimensional array that works as data container. In this sense, input and output data, obtained feature maps and convolutional filter weights can be represented as tensors. The dimensions of a tensor can be assigned with a name. A *mesh* is also a multidimensional array, however, unlike the tensor structure, it describes a virtual topology for the computational nodes (devices) of the platform, i.e., a structured abstraction of its physical topology. As in the tensor case, each dimension of the mesh array can be assigned with a name. Finally, a *layout* is an injection mapping from a tensor dimension to a mesh dimension. The layout describes how a tensor is partitioned across the specified dimension and distributed between the devices in the specified mesh dimension. Every tensor partition is referred as a *slice*. In this context, the user specifies the layout and its device distribution over the mesh, and hence, designs the model parallelism structure of the model training.

Mesh-TensorFlow does not provide a way to map two tensor dimensions into one mesh dimension. Furthermore, it assumes always a homogeneous platform, i.e., it requires the size of the partitioned tensor dimension to be evenly divisible by the size of the mesh dimension. Devices of the platform apply parallel computations on their assigned slices and they communicate results by using MPI reduction primitives to reduce out tensor dimensions when needed. If a layout is not specified for a tensor, such tensor is replicated in all devices.

Algorithm 1. Convolutional neural model definition

$\mathbf{X} \in \mathbb{R}^{B\times N\times M}$: Input volume
$mesh = [(\text{“}b_0\text{”}, d)]$
$f_0 = dimension(\text{“}filters\ 0\text{”}, sz_0)$
$f_1 = dimension(\text{“}filters\ 1\text{”}, sz_1)$
$f_2 = dimension(\text{“}filters\ 2\text{”}, sz_2)$
$layout = [filters0 : b_0]$
$\mathbf{Z}_0$ = conv2d($\mathbf{X}, f_0, [k_0 \times k_0]$)
$\mathbf{X}_0$ = ReLU($\mathbf{Z}_0$)
$\mathbf{Z}_1$ = conv2d($\mathbf{X}_0, f_1, [k_1 \times k_1]$)
$\mathbf{X}_1$ = ReLU($\mathbf{Z}_1$)
$\mathbf{Z}_2$ = conv2d($\mathbf{X}_1, f_2, [k_2 \times k_2]$)
$\mathbf{X}_2$ = ReLU($\mathbf{Z}_2$)
$\mathbf{Z}_3$ = FClayer($\mathbf{X}_2, fc, reduceDim = true$)
$\mathbf{X}_3$ = ReLU($\mathbf{Z}_3$)
$logits$ = FClayer($\mathbf{X}_3, number\_of\_classes$)

Algorithm 1 provides a pseudo-code of the model parallelism definition with Mesh-TensorFlow. In particular, it describes a CNN model with three convolutional layers as baseline. The input data is denoted as $\mathbf{X} \in \mathbb{R}^{B\times N\times M}$, where naturals B, N, M are the batch size and the height and width spatial dimensions, respectively (as Mesh-TensorFlow works in the spatial dimension, the spectral dimension can be omitted to simplify the mathematical nomenclature). As the data moves through the convolutional layers (denoted as conv2d), these apply their filters to the data, overlapping them to small windows of the input data (defined by the local receptive field) and extracting as a result an intermediate output feature volume. In this way, the l-th conv2d receives $\mathbf{X}_{l-1}$ as input data, applying its f_l filters individually as the element-wise product between the kernel weights $\mathbf{W}_l$ and the input elements that fall

into the local receptive field with size $k_l \times k_l$, to which the bias $\mathbf{b}_l$ is added. The resulting array $\mathbf{Z}_l$ can be interpreted as the extracted feature maps which indicates both the occurrence of that feature learned by the filter $\mathbf{W}_l$ and its location within the input data. Eq. (1a) provides the general formulation of a convolution layer:

$$\mathbf{Z}_l = \mathbf{W}_l * \mathbf{X}_{l-1} + \mathbf{b}_l \tag{1a}$$

$$\mathbf{Z}_l(i,j,t) = \sum_{\hat{i}=1}^{k_l}\sum_{\hat{j}=1}^{k_l}\mathbf{W}_l(\hat{i},\hat{j},t)\mathbf{X}_{l-1}(i+\hat{i},j+\hat{j}) + \mathbf{b}_l(t) \tag{1b}$$

Eq. (1b) specifies the extraction of the (i,j,t)-th feature within $\mathbf{Z}_l$ volume, where i,j and $\hat{i},\hat{j}$ are the spatial indices that cover the input and weights arrays, respectively, and $t = 1,\ldots,f_l$ indicates the filter index. Fig. 1 provides the graphical interpretation of these equations. In addition, to learn the non-linear patterns, a non-linear activation function (in this case the Rectified Linear Unit is implemented, i.e. $\mathbf{X}_l = \mathrm{ReLU}(\mathbf{Z}_l) = \max(0,\mathbf{Z}_l)$ [42,43]) is included, which returns the final output volume $\mathbf{X}_l \in \mathbb{R}^{f_l \times N_l \times M_l}$. In this case, as three conv2d layers are implemented, l is set to $l = 1,2,3$. Once the feature extraction stage conducted by conv2d layers is completed, the obtained data representation is processed by a multilayer perceptron (MLP), which comprises two fully-connected layers, denoted as FClayers. This MLP is the final classifier, where the FClayers contain *fc* and *number_of_classes* perceptrons respectively.

Furthermore, *mesh* describes a one-dimensional virtual topology, which is denoted as b_0 and is composed by d devices, defining the tuple ("b_0", d). Then, *dimension* variables f_0,f_1 and f_2 are defined, where each one specifies the names ("*filters*1", "*filters*2" and "*filters*3") and sizes (sz_1, sz_2 and sz_3) for a tensor dimension. Such names are used later in the *layout* description. In turn, *layout* describes the mapping of the tensor dimension described previously (*filters*0) to a mesh dimension (b_0). In particular, such mapping is only specified for dimension f_0, which means that no partition is made for those kernel tensors in the second and third conv2d layers. As a direct consequence, in the first conv2d layer, the corresponding kernel tensor is automatically partitioned and distributed across the f_0 dimension to the d devices specified in the mesh dimension b_0. As a result, the model is partitioned in $[k_0, k_0, sz_0/d]$ slices per device. On the contrary, second and third conv2d layers are not included in the layout, hence, their kernel tensors are replicated within each device. Between conv2d layers and FC layers, Mesh-TensorFlow internally performs a MPI reduction operation to combine the tensor slices distributed in the first layer. Fig. 2 provides the graphical representation of kernel tensors at first (left), second and third (right) conv2d layers. On the left side, the kernel tensor is partitioned across the number of filters dimension (sz_0/d), while on the right side, the kernel tensor are replicated in each device.

In addition, it is noteworthy that, partitioning a feature map tensor (i.e., $\mathbf{X}$) along the batch dimension (B) is equivalent to perform data parallelism, since actually the data is partitioned between devices.

Fig. 3 shows the complete landscape of the described CNN model in Algorithm 1, considering $d = 3$ devices. Note that, although devices represented in the figure have different computational capabilities, Mesh-TensorFlow divides the kernel tensors into homogeneous slices. This homogeneous partitioning can be considered unfair, as it does not fully-exploit the more powerful devices, while it can overwhelm the less powerful ones. As a result, the processes required in each device perform calculations at different speeds, due to the different computing and memory capabilities, and hence they arrive to the reduction communication phase (just before the fully connected layers) at different times, which have a decisive impact in the overall performance. in order to overcome this limitation, this paper introduces the HetMesh-Tensorflow approach as a new strategy to perform heterogeneous partitioning, taking into account the devices' capabilities.

4. Hetmesh-Tensorflow approach

This section provides the details of the proposed method. The main goal pursued by the developed model is the efficient and effective training of deep CNN-inspired models, following a model parallelism scheme based on a heterogeneous HPC platform. In this sense, and following the previous section, we assume a set of computational nodes or devices with different computational capabilities (speeds).

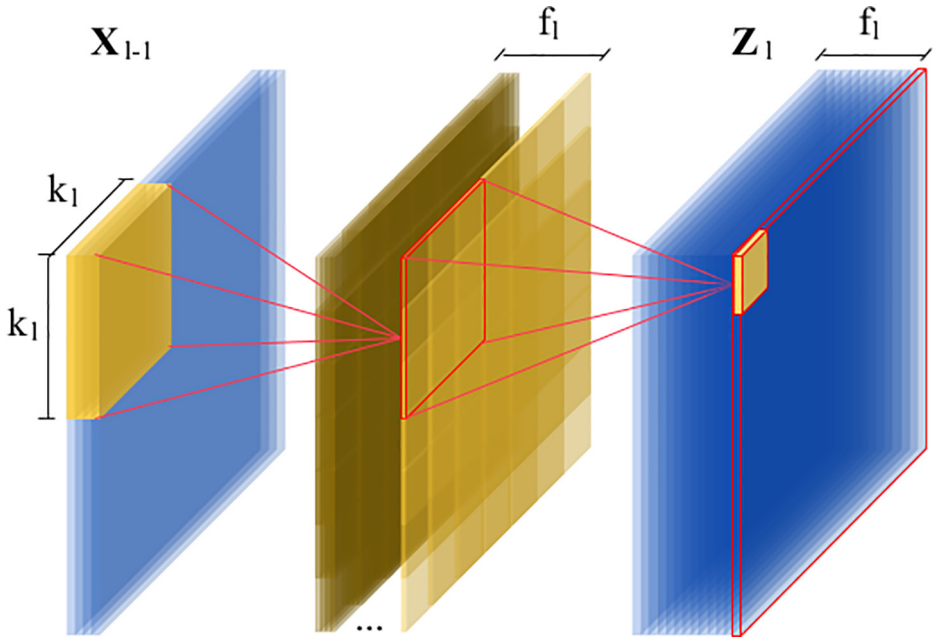


Fig. 1. Graphical representation of standard conv2d layer. f_l filters with size $k_l \times k_l$ are overlapped on the input volume $\mathbf{X}_{l-1} \in \mathbb{R}^{B \times N \times M}$, obtaining the intermediate feature maps $\mathbf{Z}_l \in \mathbb{R}^{f_l \times N_l \times M_l}$, where output spatial dimensions are obtained as $N_l = \left\lfloor \frac{N-k_l+2\rho_l}{s_l} \right\rfloor + 1$ and $M_l = \left\lfloor \frac{M-k_l+2\rho_l}{s_l} \right\rfloor + 1$ (ρ_l is the padding and s_l the stride).

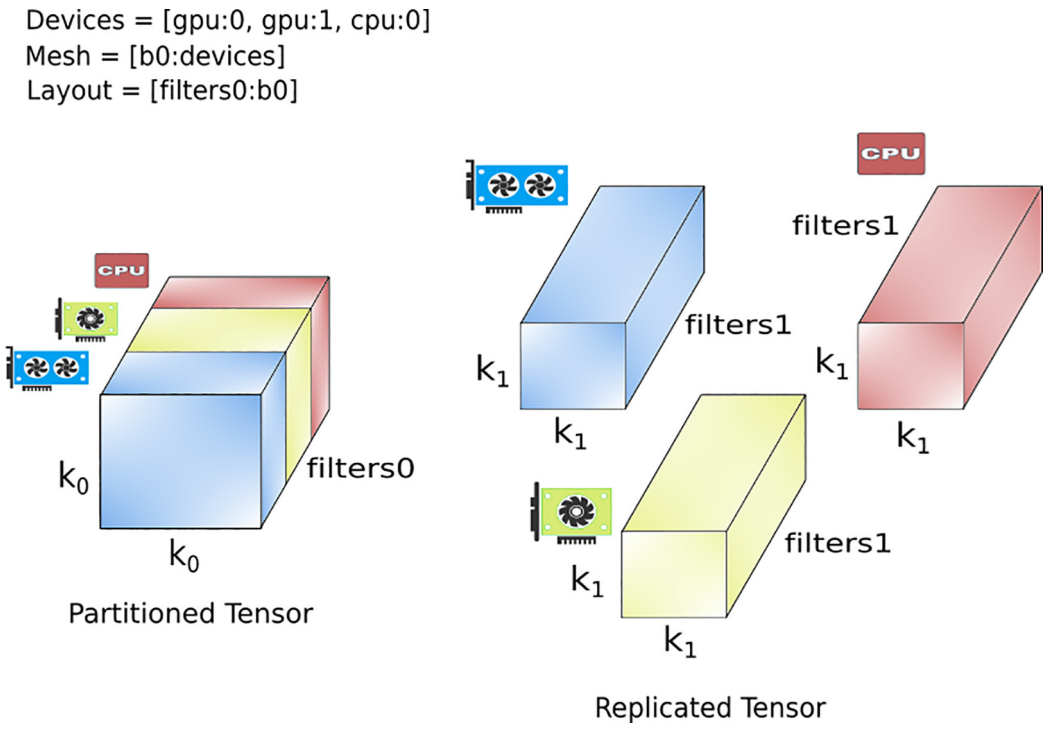


Fig. 2. Partitioning and Mapping defined by a *layout* of the kernel tensors in convolutional layers of the Algorithm 1 to a one-dimensional *mesh* with $d = 3$ devices. At the left part, kernel tensor in the first convolutional layer is shown, that is partitioned across *filter*0 dimension between available devices in three slices. At the right part, kernel tensor in the second convolutional layer is shown, that is replicated in every device. The array *devices* contain a description of the available devices in the platform. The tensor in the first convolutional layer is evenly partitioned between devices, independently of each device computational capabilities.

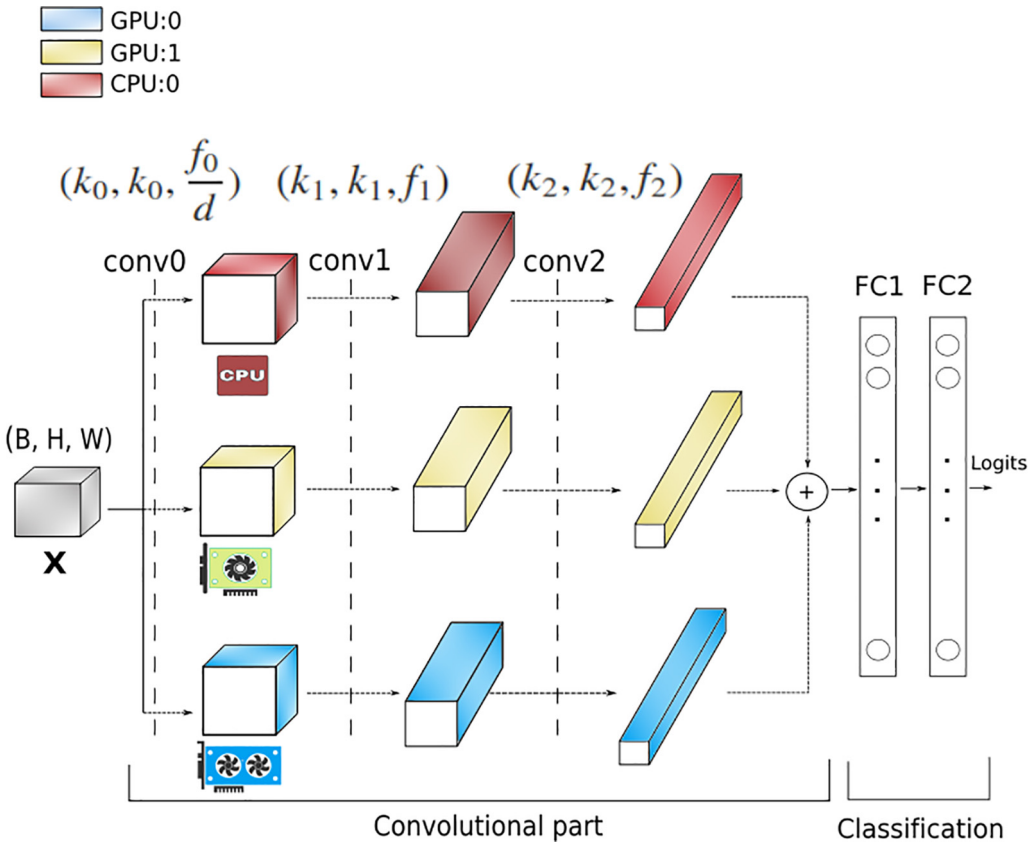


Fig. 3. Mesh-TensorFlow homogeneous partitioning of those kernel tensors defined by the three convolutional layers specified in Algorithm 1, considering $d = 3$ devices.

4.1. Homogeneous distribution issues

Original Mesh-TensorFlow performs tensor partitioning across a certain dimension in equally proportioned slices, and hence, every device is assigned with the same amount of workload, independently of its speed. As a result, faster devices will wait for the slower ones in every reduction communication point, which can lead to performance degradation.

To overcome this limitation, we propose a new methodology, assuming a heterogeneous platform composed by several devices with different computational capabilities. Our proposed method focuses on the partitioning of kernel tensors across the filter dimension considering 1D-meshes in convolutional layers. As a result, model parallelism scheme is provided.

4.2. Strategy to adapt workload to speeds

In order to perform heterogeneous partitions and distributing tensor slices between a set of heterogeneous devices depending on their capabilities, a key issue should be addressed: How can we accurately determine the speed of each device to balance the workload?.

In this regard, FuPerMod [29] is an interesting tool that has been wisely-used in scientific HPC applications to distribute the workload between a set of heterogeneous nodes. In fact, FuPerMod executes a benchmark to accurately determine the devices speeds. As a result, FuPerMod returns the relative speed of the d devices comprised by the heterogeneous platform, as a set of d functions indicated by Eq. (2):

$$\mathbf{s} = \{s_1(p), s_2(p), \ldots, s_d(p)\} \in \mathbb{R}^d, \tag{2}$$

where each function $s_i(\cdot)$ varies along a given range of task sizes p (given in bytes) to represent the device performance. The speed is obtained as the inverse of the time t_i invested in executing the benchmark for every size p_j that falls within the search range (where p indicates the maximum limit), i.e., $s_i(p_j) = p_j/t_i(p_j)$. Load balancing is achieved when $t_i \approx t_j, \forall i,j \in \{1,\ldots,d\}$. The final goal is to find $\mathbf{p} = \{p_1,\ldots,p_d\} \in \mathbb{R}^d$ s.t. $\sum_i p_i = p$, which is formulated by FuPerMod as an optimization problem and solved using an iterative geometrical method [44]. Then, the relative differences in the amount of work, which is assigned to each device and represented as vector $\mathbf{p}$, are used to obtain the optimal size of the kernel tensor slice across the filter dimension to be distributed to the devices. With this valuable information, it performs the final partitioning to balance the workload.

Both, the partition vector $\mathbf{p}$ and the vector of speeds $\mathbf{s}$ are determined statically by running FuPerMod tool in a previous step to the model training, and therefore, its computations do not affect to the training performance. In addition, it is not necessary to always run it before training, since launching it once, the results are stored. Obviously, determining the speed of a device, and hence, its assigned slice, may not be completely accurate. However, small errors that may appear are assumed as common jitters that could appear even in homogeneous systems. In this regard, Moreno et al. [20] proposed an accurate method to balance the workload to a set of processes training a CNN based on FuPerMod, under a data parallelism scheme, by using as a benchmark a CNN on a range of tensor sizes.

4.3. Heterogeneous partitioning

Once the speeds of the devices are determined by [20], tensors are partitioned across the dimension specified in the layout to the virtual 1D-mesh of devices. Algorithm 1 remains unchanged, because the partitioning step is completely transparent to the user.

The tensor will be only partitioned once in a convolutional layer according to both the tensor filters dimension and vector speeds $\mathbf{s}$. The dimension specified in the layout will determine the number of partitions for the following convolutional layers. Fig. 4 provides a graphical representation of this procedure. Note the difference with the previous Fig. 3, where the first conv2d was homogeneously partitioned between devices. In particular, in Fig. 4 the partitioning within first conv2d layer filters can be observed, where the f_0 filters of size $k_0 \times k_0$ are distributed among the different devices as $f_0^0 \neq f_0^1 \neq f_0^2$ and s.t. $f_0 = f_0^0 + f_0^1 + f_0^2$, according to vector $\mathbf{s}$.

As pointed before, CNNs are divided into two parts: the feature extractor net and classification net. The feature extractor net is composed by the set of convolutions and pooling layers, obtaining as a result the feature maps where the existence and position of the features learned by the filters are indicated. According to Mesh-TensorFlow implementation, in purely convolutional partitioned layers, the workload that each device will have to process (i.e. the slice assigned to the device) is $(B, N_l, M_l, |f_l^i|)$, being f_l^i the number of filters of a kernel tensor in the l-th convolutional layer assigned to the i-th device, with $i = 1,\ldots,d$, while N_l and M_l are the respective height and width of the output feature volume $\mathbf{X}_l$ and B the batch size. In particular, those filters assigned to a device f_l^i are computed using FuPerMod as described in Section 4.2.

Once a tensor is partitioned according to the layout, the model will not replicate again until reach the classification net, implemented by a MLP. This slightly changes the behavior with respect to Mesh-TensorFlow (as detailed in Section 4.4), in which partitioning is performed repeatedly along all convolutional layers in forward propagation.

Tensor slices must be stacked at every communication point (reduction) in order to compose the tensors and update them during the backward propagation stage, by using the computed gradients. Thus, to reduce out the parameter slices, all dimensions except the filter dimension must match in all slices.

4.4. Semantics of proposed tensor partitioning

Original Mesh-TensorFlow has several problems when partitioning over the filter dimension for multiple convolutions, in addition to its unfair partitioning procedure in heterogeneous environments.

As described in Section 3, the layout indicates the tensor dimensions to be partitioned. Therefore, we can define the total number of filter dimensions to be partitioned as n. The mesh indicates the number of devices to be assigned over a dimension. The first problem when facing multiple convolutions is that the $mesh - layout$ relation produces an increment in the number of partitions over the mesh dimensions, with a total of d^n partitions.

To understand this problem, we first define the Eqs. (1a) and (1b) that describe the standard application of the l-th convolutional layer on the input data $\mathbf{X}_{l-1} \in \mathbb{R}^{B_l \times N \times M}$ (where $B_l = f_{l-1}$ indicates the input channels), by convolving the weight matrix $\mathbf{W}_l \in \mathbb{R}^{f_l \times k_l \times k_l \times B_l}$ which results into the output volume $\mathbf{Z}_l \in \mathbb{R}^{f_l \times N \times M}$ (for simplification, we keep constant the spatial dimension). Therefore, f_l convolution filters are overlapped onto small patches of the input, particularly with receptive field size $k_l \times k_l \times B_l$, and conduct matrix calculations to obtain for each application the corresponding output feature. In this sense, a CNN model with L convolutional layers has to train a number of parameters given by Eq. (3a), involving a high number of floating point operations (FLOPs):

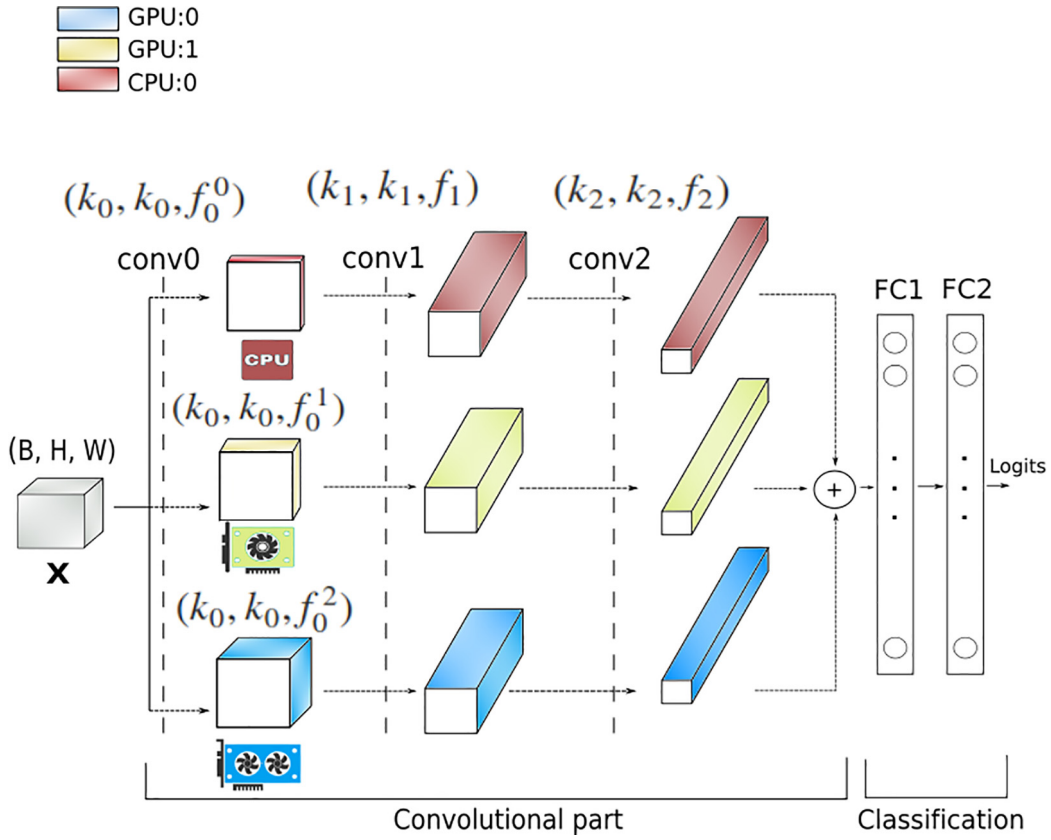


Fig. 4. Graphical representation of heterogeneous kernel partition (according to vector speeds **s**) within first convolutional layer, and standard kernel replication within second and third layers, following the CNN model proposed in Algorithm 1, considering $d = 3$ devices that are represented in different colors.

$$\text{Parameters}: \sum_l k_l \times B_l \times f_l \tag{3a}$$

$$\text{FLOPs}: \sum_l f_l \times (N \times M) \times B_l \times k_l^2 \tag{3b}$$

In this regard, the number of filters in the current layer f_l has a decisive impact on both the number of parameters (increasing storage requirements) and the number of FLOPs executed (increasing the computational burden). In this sense, the proposal divides the l-th layer across the filter domain, grouping the f_l filters into d groups, where d corresponds to the number of available devices. Accordingly, each device receives f_l/d filters, which significantly reduces both computational load and memory consumption.

Hence, focusing on the base-line model proposed by Figs. 3–5, where a 3-layer CNN model is depicted, it is possible to observe how the proposal divides each convolutional layer between 3 devices. As a consequence, 9 cropped-convolutions are obtained and distributed into the devices. As the proposal maintains this partitioning throughout the execution, further partitioning of the layers will imply a new division to each of the nine chunks. As a result, 27 cropped-convolutions would be obtained. This exponential growth generates a large number of inefficient partitions, as they involve an undesirable increment in communications.

The second problem is related to the limitation that two contiguous convolutions cannot be partitioned over the filter dimension. This is because the partitioned filters of the contiguous convolution would be applied on an input already partitioned, resulting in a significant loss of information.

Algorithm 2. Algorithm for heterogeneous slices partitioning.

1: d, total number of devices
2: $pnum = 0 \dots d-1$, the device identifier
3: $filters_{layer}$, total filters comprised by the layer
4: $sliceShape^{pnum}$, vector where each element indicates filters per device
5: s = $[s^0, s^1, \dots, s^{d-1}]$, the speeds array
6: sliceShape = $[k, k]$, the shape of the slice, where k is the kernel size
7: sliceBegin = $[0, 0]$, the start of each slice in the convolution
8: **for** $pnum = 0, 1, \dots, d-1$ **do**
9: **for** *layer* **in** *layoutLayers* **do**
10: **if** *layer* **is** $evenConvolution$ **then**
11: $layerShape^{pnum}$ = toInteger($s^{pnum} \times filters_{layer}$)
12: sliceShape.append($layerShape^{pnum}$)
13: **if** *pnum* **is** 0 **then**
14: begin = 0
15: **else**
16: begin = $\sum_{pnum=0}^{d-1} layerShape^{pnum}$
17: **end if**
18: sliceBegin.append(begin)
19: **else if** *layer* **is** $oddConvolution$ **then**
20: sliceShape.append($filters_{layer}$)
21: sliceBegin.append(0)
22: **end if**
23: **end for**
24: **end for**

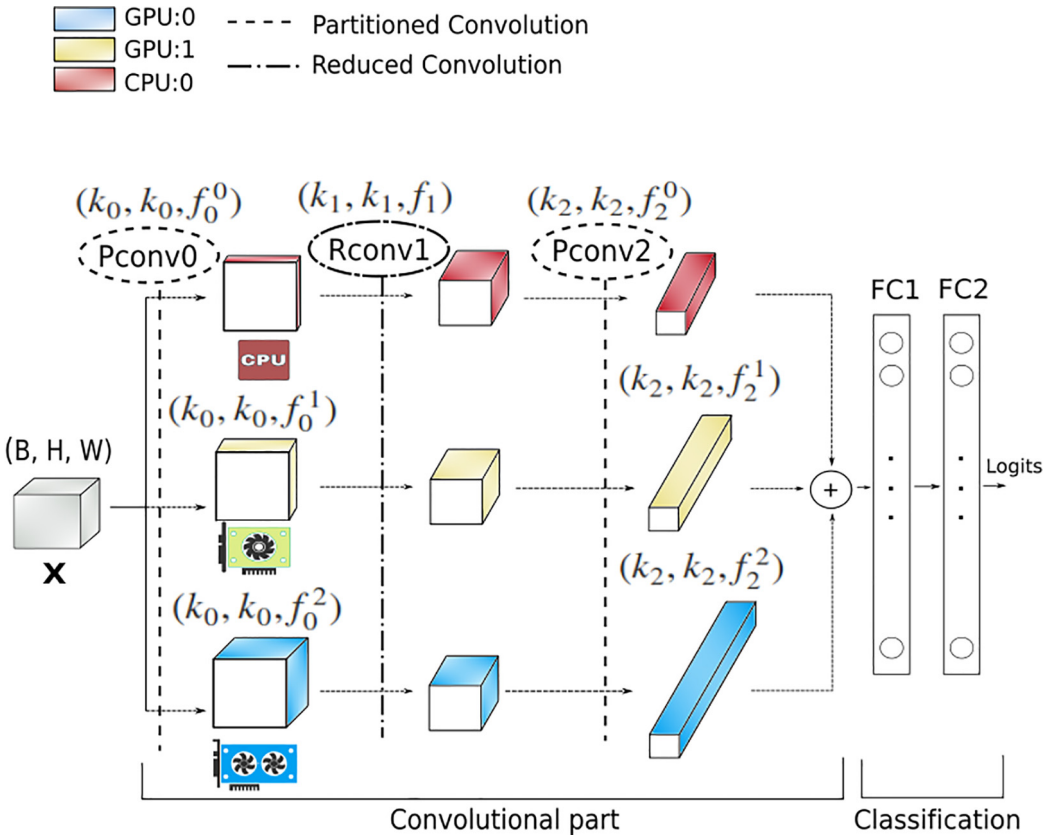


Fig. 5. Graphical representation of the heterogeneous partitioning to solve Mesh-TensorFlow filter partitioning problems. The loss of features problem is solved using two partitioning convolutions *Pconv* and one reduce convolution *Rconv* in the middle of both. Also, the one-dimensional mesh extension is used.

To overcome both problems, two solutions have been implemented over the original proposed method. First, our heterogeneous proposal changes the layout semantic established by Mesh-TensorFlow at some points. In this sense, there is still a mesh which represents the virtual topology of the devices in the heterogeneous platform and the layout is still composed by pairs *(tensor-dimension, mesh-dimension)*, however, only the number of devices on the first dimension of filters to be partitioned will be indicated in the mesh. Subsequently, the mesh will be replicated for the following partitions on the filters. Hence, taking into account the model represented in Fig. 4 and described in Table 1, the mesh and the layout from Algorithm 1 are adapted to $mesh = [(\text{“}b_0\text{”}, d), (\text{“}b_1\text{”}, 1), (\text{“}b_2\text{”}, 1)]$ and $layout = [filters0 : b_0, filters1 : b_1, filters2 : b_2]$, respectively.

To get over the loss of features, two types of convolutional layers have been defined when partitioning across the filter dimension. On the one hand, the partitioning convolutions, also called *even* convolutions *Pconv* are defined as the building block in which both workload partitioning and distribution are made between the devices across the filter dimension. On the other hand, the *odd* convolutional layers, also called reduce convolutions *Rconv* are the building blocks that receive and process a partition from the previous *Pconv* convolution. Thus, in *Rconv* all filters must be processed in each device in order to preserve the model precision, avoiding the loss of information. The complete behavior of these two types of conv2d layers can be observed on Fig. 5 where both are represented with its respective filters assignment per layer and device. There is a notable difference with the previous Fig. 4 which refers to the original Mesh-TensorFlow code and hence, only one layer can be partitioned to avoid the two problems described in this Section. A simple vision for the heterogeneous partitioning with these changes is showed in Algorithm 2.

With these two modifications, it has been possible to preserve the efficiency provided by the model parallelism given by Mesh-TensorFlow, adapting it to a heterogeneous partitioning across filter dimension.

Table 1
Layers of the convolutional neural network implemented for classification of the MNIST dataset.

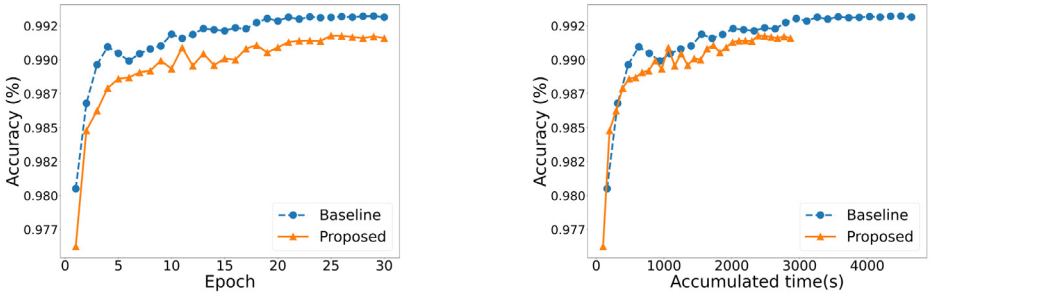
Model for MNIST		
Layer ID	Kernel/Neurons	Activation Funct.
Conv0	$7 \times 7 \times 60$	ReLU
Conv1	$5 \times 5 \times 120$	ReLU
Conv2	$3 \times 3 \times 180$	ReLU
Avg-Pool	2×2	–
FC1	128	ReLU
FC2	$n_{classes}$	Softmax

4.5. Performance analysis

As explained before, the workload of each convolution will be determined by the computational capacity of each device (collected by vector **s**) with the number of filters. Combining the devices fairly in the processing of convolutional kernels can increase the performance of the deep model. In this sense, the performance is determined by the computation and communication times. Therefore, focusing on both homogeneous and heterogeneous partitioning, the computational and communication cost is indicated by Eq. (4):

$$t_{comp} = \max(B \cdot z_{io} \cdot c_u^i) \tag{4a}$$

$$t_{comm} = B \cdot z_{io} \tag{4b}$$



a) Accuracy per epoch. b) Accuracy with accumulate time over training.

Fig. 6. Model described in Table 1 trained for 30 epochs in the first machine.

where B denotes the batch size, $c^i = [k_l, k_l, f_l^i]$ defines the local receptive field and the number of filters (per device i) of the l-th convolutional layer, z_{io} represents the units for the input and output layers and u the neurons units.

The computation time t_{comp} will be given by the slowest device. This is because to avoid staleness synchronous communication is used. As for the heterogeneous approach the workload balancing has previously occurred, there is no significant difference between the maximum time and the average time of all devices. Conversely, Eq. (4a) under the homogeneous approach proposed by Mesh-TensorFlow is significantly higher due to waiting times between devices. This is analyzed in Section 5.

5. Experimentation and analysis

In this section the heterogeneous approach is evaluated in terms of performance compared to the homogeneous distribution. The platform is detailed, and then we discuss the results obtained in the training phase in terms of performance and accuracy. The experimentation is divided into two parts. First, a basic network is used to demonstrate the behavior of the proposal in a simpler way. Subsequently, more extensive tests have been performed with deeper networks and more complex datasets.

Tests have been carried out in a single machine environment because the distributed model parallelism from Mesh-TensorFlow implementation is under development and currently, it does not support multiple node platforms.

To obtain initial insights of the implementation, the first machine used is an X Generation Intel Core i9-9940X processor with 19.25M of Cache and up to 4.40 GHz (14 cores/28 way multi-task processing). Motherboard is a Gigabyte X299 Aorus with 128 GB of DDR4 RAM. Graphic processing units are a NVIDIA GeForce GTX 2080Ti with 11 GB GDDR6 of video memory and 4352 cores, and a NVIDIA Titan RTX with 24 GB GDDR6 of video memory and 4608 cores. On the other hand, the second machine used belongs to the Ceta-Ciemat supercomputing cluster. This cluster provides with a more realistic environment composed of four NVIDIA Volta V100 with 32 GB RAM HBM2 of video memory and 5120 cores. The processor is an Intel Xeon Gold 6240 with 196 GB RAM and 24.75M cache and 18-cores up to 3.90 GHz. In order to evaluate this proposal, all the computing elements are used in the first machine, while 3 GPUs and CPU are used for the *Ceta-Ciemat* machine.

Models were trained using two standard datasets, MNIST and CIFAR100. MNIST is composed of black and white handwritten digits images of size $28 \times 28 \times 1$, representing digits from 0 to 9, with a training set of 60.000 examples and a test set of 10.000 examples. CIFAR100 contains 60.000 colour images of size $32 \times 32 \times 3$ of 100 non-biased classes. This dataset is used to verify that the proposed implementation provides successful results in a complex classification problem with a high number of classes.

Table 2
Results for time and accuracy under model described in Table 1.

		Baseline	Proposed
Epoch time	Min.	154.27	**93.57**
	Max.	160.82	**100.91**
	Avg.	155.06	**95.54**
Accuracy		**99.32**	99.17

Numerous models are used in the experiments. First, we use the model shown in Table 1 to get the first approximation test of the HetMesh implementation efficiency in a heterogeneous system. This model is composed of a convolutional part of 3 convolutional layers and a average-pool layer. The classification part is composed of 2 fully-connected layers and the softmax.

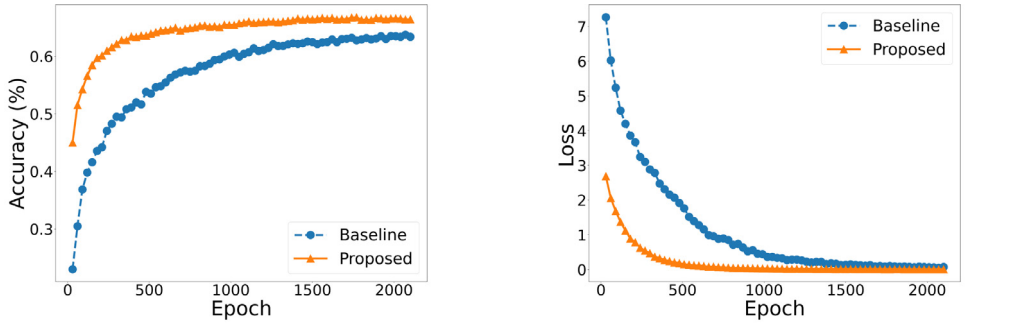
Each model is executed five times using both the homogeneous and the heterogeneous distributions. The results in the figures show the mean of all runs along with the standard deviation.

Note that the gain obtained from performing the heterogeneous distribution over the homogeneous one will be conditioned by the difference between the devices (both GPU and CPU) where the training is executed.

The evolution of the precision across the training step has been studied. The main objective of the heterogeneous distribution is to be time efficient without losing precision. Fig. 6(a) shows the evolution of the precision along epochs for both proposals. As we can see, our proposal accuracy percentage is almost the same as the original homogeneous throughout the training. Furthermore, Fig. 6(b) shows a clearly advantage of the heterogeneous proposal, that completes the training significantly earlier. The results shown in the Fig. 6 are detailed in the Table 2.

Once the efficiency of the heterogeneous proposal was introduced and verified, tests were carried out with deeper CNN models using the *Ceta-Ciemat* machine. These networks are the well-known VGG11, VGG16 and VGG19. In these VGG networks, the model is evaluated every 30 epochs, and therefore, the shuffling of the data occurs at that time.

At this point, it is worth noting the accuracy gap observed in Fig. 7(a) between our proposal and the original implementation of partitioned convolutions. Indeed, the obtained results are in line with recently developed theories in the deep learning field as described below. As the convolution layer attempts to learn and adjusts its filter weights in a 3D space (width, height and channel dimensions), each filter has to simultaneously map the correlations between channels and spatial dimensions [45], which may result in a correlation between different filters in the same convolutional layer [46]. In this respect, it has been observed that dividing the filters into several convolutions and then combining them rather



a) Accuracy per epoch. b) Training loss adjustment over training.

Fig. 7. VGG16 model trained for 2100 epochs on the *Ceta-Ciemat* platform.

Table 3
Time (seconds) per epoch for VGG models.

		Baseline	Proposed
VGG11	Min.	5402.21	**922.61**
	Max.	5460.87	928.25
	Avg.	5419.24	924.72
	Speedup		**5.86**
VGG16	Min.	9619.58	**1943.50**
	Max.	9681.67	1960.34
	Avg.	9657.80	1953.91
	Speedup		**4.94**
VGG19	Min.	13135.04	**2123.94**
	Max.	13255.21	2165.82
	Avg.	13218.72	2147.32
	Speedup		**6.15**

Table 4
Accuracy results for VGG models.

		Baseline	**Proposed**
VGG11	Max.	61.26	**63.75**
	Last.	61.16	63.71
VGG16	Max.	63.74	**66.79**
	Last.	63.71	66.43
VGG19	Max.	61.59	**66.61**
	Last.	61.30	66.61

than applying them all in one convolutional layer is more computationally efficient and provides better precision results [47]. Although the black box nature of the convolution layer prevents the accurate interpretation of the extracted features, recent works find that dividing the full convolution layer into smaller convolutions (or even finding a smaller set of basis filters) reduces model parameters, prevents model overfitting and reduces the correlation between kernel filters [47,45,48,46,49,50]. In this sense, the proposed heterogeneous model divides the odd-layer filters into parallel convolutions and the independently extracted features are concatenated and combined in the even-layers. On the contrary, the homogeneous model applies standard convolutions in all its layers, except for the last one, which is divided homogeneously. Therefore, obtained results seem to indicate that the heterogeneous model is able to extract less redundant data representations, which in the end helps to improve the accuracy results. Furthermore, at Fig. 7(b) it is observed that our heterogeneous proposal has adjusted the loss more than the homogeneous original proposal in a significantly less amount of time.

Regarding the training time, the results in the Table 3 shows the epoch training times of these methods. It is appreciated how there is a notable decrease in the epoch times when more convolutions are partitioned in a heterogeneous way. Thus, it is demonstrated that our proposal in this article achieves a notable improvement in training times by being able to heterogeneously partition all convolutions (see Table 4).

As shown in the Table 3, the speedup depends on the number of convolutions that are partitioned. It is important to remember that only even convolutions *Pconv* are partitioned. Therefore, the speedup depends on how many convolutions with greater computational load are denoted as *Pconv*.

6. Conclusion

A new approach to distribute the training of deep networks on dedicated heterogeneous platforms is described. To achieve this, the computing capabilities of each of the devices must be determined in a previous step. With this, the training platform is characterized. This provides a computational load balance through the model parallelism scheme. Specifically, this computational adjustment is made over the model convolution filters. Thus, waiting times at communication points are almost eliminated. These synchronization points are located in those layers in which the model is not partitioned. This causes a reduction of the overall execution time needed for training the deep model as demonstrated for two different platforms and datasets. Furthermore, our heterogeneous partitioning proposal has no negative impact on model precision while reducing the training time. Thus, it increases the convergence of deep models, while providing improved accuracy.

One of the main contributions of this work is the exploitation of well-known HPC optimization techniques to balance the distributed training of deep learning models on model parallelism schemes. Previous works focus on the homogeneous distribution of the workload between devices, assuming a homogeneity between the devices. Conversely, our work takes into account the computation differences between devices, being able to adapt the computation made on each of them.

This development could be extended in terms of scability by increasing the number of replicas, using larger datasets and different devices. The proposed HPC techniques used for the implementation of this proposal will greatly facilitate the increase of replicas. This is the main reason why it has been decided to use this type of techniques. Hence, scalability test will be carried out as soon as the distributed model parallelism implementation is provided by Mesh-TensorFlow.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

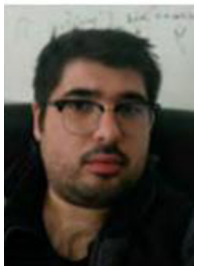
Supported by (1) The European Regional Development Fund 'A way to achieve Europe' (ERDF) and the Extremadura Local Government (Ref. IB16118); (2) The Spanish Ministry of Science and Innovation (Ref. PID2019-110315RB-I00 APRISA); and (3) The computing facilities of Extremadura Research Centre for Advanced Technologies (CETA-CIEMAT), funded by the European Regional Development Fund (ERDF). CETA-CIEMAT belongs to CIEMAT and the Government of Spain.

References

[1] I. Goodfellow, Y. Bengio, A. Courville, Deep Learning, MIT Press, 2016.
[2] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition, Proceedings of the IEEE 86 (11) (1998) 2278–2324.
[3] O.I. Abiodun, A. Jantan, A.E. Omolara, K.V. Dada, N.A. Mohamed, H. Arshad, State-of-the-art in artificial neural network applications: a survey, Heliyon 4 (11) (2018) e00938.
[4] A. Esteva, A. Robicquet, B. Ramsundar, V. Kuleshov, M. DePristo, K. Chou, C. Cui, G. Corrado, S. Thrun, J. Dean, A guide to deep learning in healthcare, Nature Medicine 25 (1) (2019) 24–29.
[5] M. Paoletti, J. Haut, J. Plaza, A. Plaza, Deep learning classifiers for hyperspectral imaging: a review, ISPRS Journal of Photogrammetry and Remote Sensing 158 (2019) 279–317.
[6] Y. LeCun, Y. Bengio, G. Hinton, Deep learning, Nature 521 (7553) (2015) 436–444.
[7] N. O'Mahony, S. Campbell, A. Carvalho, S. Harapanahalli, G.V. Hernandez, L. Krpalkova, D. Riordan, J. Walsh, Deep learning vs. traditional computer vision, in: Science and Information Conference, Springer, 2019, pp. 128–144..
[8] Z. Wang, J. Chen, S.C. Hoi, Deep learning for image super-resolution: a survey, IEEE Transactions on Pattern Analysis and Machine Intelligence..
[9] Y. LeCun, Deep Learning, Hardware: past, present, and future, in: IEEE International Solid-State Circuits Conference-(ISSCC), IEEE, 2019, pp. 12–19.
[10] Q.V. Le, J. Ngiam, A. Coates, A. Lahiri, B. Prochnow, A.Y. Ng, On optimization methods for deep learning, in: Proceedings of the 28th International Conference on Machine Learning, ICML'11, Omnipress, USA, 2011, pp. 265–272, ISBN 978-1-4503-0619-5..
[11] A. Sergeev, M.D. Balso, Horovod: fast and easy distributed deep learning in TensorFlow, CoRR abs/1802.05799..
[12] J. Dean, G.S. Corrado, R. Monga, K. Chen, M. Devin, Q.V. Le, M.Z. Mao, M. Ranzato, A. Senior, P. Tucker, K. Yang, A.Y. Ng, Large scale distributed deep networks, in: NIPS, USA, 2012, pp. 1223–1231..
[13] P. Goyal, P. Dollár, R. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, K. He, Accurate, Large minibatch SGD: training ImageNet in 1 hour, 2017..
[14] X. Jia, S. Song, W. He, Y. Wang, H. Rong, F. Zhou, L. Xie, Z. Guo, Y. Yang, L. Yu, T. Chen, G. Hu, S. Shi, X. Chu, Highly scalable deep learning training system with mixed-precision: training ImageNet in four minutes, 2018..
[15] C. Ying, S. Kumar, D. Chen, T. Wang, Y. Cheng, Image classification at supercomputer scale, 2018..
[16] F. Seide, H. Fu, J. Droppo, G. Li, D. Yu, 1-bit stochastic gradient descent and its application to data-parallel distributed training of speech dnns, in: Fifteenth Annual Conference of the International Speech Communication Association, 2014.
[17] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen, Y. Zou, DoReFa-Net: training low bitwidth convolutional neural networks with low bitwidth gradients, 2016..
[18] Y. Huang, Y. Cheng, D. Chen, H. Lee, J. Ngiam, Q.V. Le, Z. Chen, GPipe: efficient training of giant neural networks using pipeline parallelism, CoRR abs/1811.06965..
[19] C. Chen, Q. Weng, W. Wang, B. Li, B. Li, Fast distributed deep learning via worker-adaptive batch sizing, in: Proceedings of the ACM Symposium on Cloud Computing, SoCC '18, ACM, New York, USA, 2018, pp. 521–521, ISBN 978-1-4503-6011-1..
[20] S. Moreno-Álvarez, J.M. Haut, M.E. Paoletti, J.A. Rico-Gallego, J.C. Díaz-Martín, J. Plaza, Training deep neural networks: a static load balancing approach, The Journal of Supercomputing ISSN 1573-0484, 10.1007/s11227-020-03200-6, URL: https://doi.org/10.1007/s11227-020-03200-6..
[21] N. Shazeer, Y. Cheng, N. Parmar, D. Tran, A. Vaswani, P. Koanantakool, P. Hawkins, H. Lee, M. Hong, C. Young, R. Sepassi, B. Hechtman, Mesh-TensorFlow: deep learning for supercomputers, in: Neural Information Processing Systems, 2018..
[22] K. Simonyan, A. Zisserman, Very deep convolutional networks for large-scale image recognition, arXiv preprint arXiv:1409.1556..
[23] Q. Zhang, M. Zhang, T. Chen, Z. Sun, Y. Ma, B. Yu, Recent advances in convolutional neural network acceleration, Neurocomputing 323 (2019) 37–51, https://doi.org/10.1016/j.neucom.2018.09.038, ISSN 0925-2312, URL: http://www.sciencedirect.com/science/article/pii/S0925231218311007.
[24] T. Ben-Nun, T. Hoefler, Demystifying parallel and distributed deep learning: an in-depth concurrency analysis, CoRR abs/1802.09941..
[25] A. Krizhevsky, One weird trick for parallelizing convolutional neural networks, ArXiv abs/1404.5997..
[26] J. Jiang, B. Cui, C. Zhang, L. Yu, Heterogeneity-aware distributed parameter servers, in: Proceedings of the 2017 ACM International Conference on Management of Data, SIGMOD '17, ACM, NY, USA, 2017, pp. 463–478, ISBN 978-1-4503-4197-4..
[27] Y. Ming, Y. Zhao, C. Wu, K. Li, J. Yin, Distributed and asynchronous Stochastic Gradient Descent with variance reduction, Neurocomputing 281 (2018) 27–36, https://doi.org/10.1016/j.neucom.2017.11.044, ISSN 0925-2312, URL: http://www.sciencedirect.com/science/article/pii/S0925231217318039.
[28] S. Moreno-Álvarez, J. Haut, M. Paoletti, J. Rico-Gallego, J. Martín, J. Plaza, Training deep neural networks: a static load balancing approach, The Journal of Supercomputing 10.1007/s11227-020-03200-6..
[29] D. Clarke, Z. Zhong, V. Rychkov, A. Lastovetsky, FuPerMod: A framework for optimal data partitioning for parallel scientific applications on dedicated heterogeneous HPC platforms, in: Parallel Computing Technologies, Springer, Berlin Heidelberg, Berlin, Heidelberg, 2013, pp. 182–196, ISBN 978-3-642-39958-9..
[30] A. Krizhevsky, I. Sutskever, G. Hinton, ImageNet classification with deep convolutional neural networks, Neural Information Processing Systems 25, doi: 10.1145/3065386..
[31] M. Shoeybi, M. Patwary, R. Puri, P. LeGresley, J. Casper, B. Catanzaro, Megatron-LM: training multi-billion parameter language models using model parallelism, 2019..
[32] S. Rajbhandari, J. Rasley, O. Ruwase, Y. He, ZeRO: Memory optimizations toward training trillion parameter models, 2019..
[33] A. Mirhoseini, H. Pham, Q.V. Le, B. Steiner, R. Larsen, Y. Zhou, N. Kumar, M. Norouzi, S. Bengio, J. Dean, Device placement optimization with reinforcement learning, 2017..
[34] A. Harlap, D. Narayanan, A. Phanishayee, V. Seshadri, N. Devanur, G. Ganger, P. Gibbons, PipeDream: fast and efficient pipeline parallel DNN training, 2018..
[35] A. Prieto, B. Prieto, E.M. Ortigosa, E. Ros, F. Pelayo, J. Ortega, I. Rojas, Neural networks: an overview of early research, current frameworks and new challenges, Neurocomputing 214 (2016) 242–268, https://doi.org/10.1016/j.neucom.2016.06.014, ISSN 0925-2312, URL: http://www.sciencedirect.com/science/article/pii/S0925231216305550.
[36] A. Sergeev, M.D. Balso, Horovod: fast and easy distributed deep learning in TensorFlow, 2018b..
[37] J. Dean, G.S. Corrado, R. Monga, K. Chen, M. Devin, Q.V. Le, M.Z. Mao, M. Ranzato, A. Senior, P. Tucker, K. Yang, A.Y. Ng, Large scale distributed deep networks, in: Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1, NIPS'12, Curran Associates Inc., Red Hook, NY, USA, 2012, pp. 1223–1231..
[38] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, S. Chintala, PyTorch: An Imperative Style, High-Performance Deep Learning Library, in: H. Wallach, H. Larochelle, A. Beygelzimer, F. d Alché-Buc, E. Fox, R. Garnett (Eds.), Advances in Neural Information Processing Systems 32, Curran Associates Inc, 2019, pp. 8024–8035, URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf..
[39] S. Sun, X. Liu, EC-DNN: A new method for parallel training of deep neural networks, Neurocomputing 287 (2018) 118–127, https://doi.org/10.1016/j.neucom.2018.01.072, ISSN 0925-2312, URL: http://www.sciencedirect.com/science/article/pii/S092523121830105X.
[40] G. Shi, J. Zhang, C. Zhang, J. Hu, A distributed parallel training method of deep belief networks, Soft Computing 24, doi: 10.1007/s00500-020-04754-6..
[41] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, X. Zheng, TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems, URL: http://download.tensorflow.org/paper/whitepaper2015.pdf, 2015..
[42] V. Nair, G.E. Hinton, Rectified linear units improve restricted boltzmann machines, in: ICML, 2010..
[43] B. Xu, N. Wang, T. Chen, M. Li, Empirical evaluation of rectified activations in convolutional network, arXiv preprint arXiv:1505.00853..
[44] D. Clarke, A. Lastovetsky, V. Rychkov, Dynamic Load Balancing of Parallel Computational Iterative Routines on Platforms with Memory Heterogeneity, in: Europar 2010/ Heteropar'2010, Lecture Notes in Computer Science, vol. 6586, Springer, Ischia-Naples, Italy, 2011, pp. 41–50.
[45] F. Chollet, Xception: Deep learning with depthwise separable convolutions, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2017, pp. 1251–1258.
[46] A.G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, H. Adam, Mobilenets: Efficient convolutional neural networks for mobile vision applications, arXiv preprint arXiv:1704.04861..

[47] R. Zhao, W. Luk, Learning grouped convolution for efficient domain adaptation., arXiv preprint arXiv:1811.09341..
[48] T.K. Lee, W.J. Baddar, S.T. Kim, Y.M. Ro, Convolution with logarithmic filter groups for efficient shallow CNN, in: International Conference on Multimedia Modeling, Springer, 2018, pp. 117–129..
[49] Y. Li, S. Gu, L.V. Gool, R. Timofte, Learning filter basis for convolutional neural network compression, in: Proceedings of the IEEE International Conference on Computer Vision, 2019, pp. 5623–5632.
[50] M. Wang, B. Liu, H. Foroosh, Factorized convolutional neural networks, in: Proceedings of the IEEE International Conference on Computer Vision Workshops, 2017, pp. 545–553.

Sergio Moreno Alvarez completed the B.Sc and M.sc. Degree in Computer Engineering at the University of Extremadura in 2017 and 2019, respectively. Currently, since October 2019 he is completing his Ph.D in the Department of Computer Systems Engineering and Telematics. As research experience, he has participated regional projects. His main interests are high performance computing, neural networks and DL. He is currently a Researcher in the School of Technology (University of Extremadura). He has published 4 JCR papers in international journals and 3 presentations at international and national conferences.



Juan Mario Haut is an Associated Professor with the Department of Communication and Control Systems at the National Distance Education University, Madrid, Spain. Also, he was a member of the Hyperspectral Computing Laboratory (HyperComp) at the Department of Technology of Computers and Communications, University of Extremadura, where he received the B.Sc and M.Sc. degrees in computer engineering in 2011 and 2014, respectively, and the Ph.D. degree in Information Technology in 2019 supported by an University Teacher Training Programme from the Spanish Ministry of Education.Dr. Haut was a recipient of the Outstanding Ph.D. Award at the University of Extremadura in 2019. His research interests include remote sensing data processing and high dimensional data analysis, applying machine (deep) learning and cloud computing approaches. In this sense, he has authored/co-authored more than 30 JCR journal articles (more than 20 in IEEE journals) and 20 peer-reviewed conference proceeding papers. Some of his contributions have been recognized as hot-topic publications for their impact on the scientific community. Also, he was a recipient of the Outstanding Paper Award in the 2019 IEEE WHISPERS conference. From his experience as a reviewer, it is worth mentioning his active collaboration in more than 10 scientific journals, such as the IEEE Transactions on Geoscience and Remote Sensing, IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing, and IEEE Geoscience and Remote Sensing Letters, being awarded with the Best Reviewer recognition of the IEEE Geoscience and Remote Sensing Letters in 2018. Furthermore, he has guest-edited three special issues on hyperspectral remote sensing for different journals. He is also an Associate Editor of the IEEE Gesocience and Remote Sensing Letters and IEEE Journal on Miniaturization for Air and Space Systems.

Mercedes E. Paoletti received the B.Sc and M.Sc. degrees in computer engineering from the University of Extremadura, C\'{a}ceres, Spain, in 2014 and 2016, respectively. Also, she obtained her PhD degree in 2020 supported by an University Teacher Training Programme from the Spanish Ministry of Education, as a member of the Hyperspectral Computing Laboratory (HyperComp) at the Department of Technology of Computers and Communications, University of Extremadura. She is currently a researcher at the Department of Computer Architecture, University of Málaga. Her research interests include remote sensing and analysis of very high spectral resolution with the current focus on DL and high performance computing.She has served as a reviewer for the IEEE Transactions on Geoscience and Remote Sensing and IEEE Geoscience and Remote Sensing Letters, in which she was recognized as a best reviewer in 2019. She was also a recipient of the 2019 Outstanding Paper Award recognition in the IEEE WHISPERS 2019 conference, and a recipient of the Outstanding Ph.D. Award at the University of Extremadura in 2020.

Juan-Antonio Rico-Gallego received the Computer Science Engineering degree and the PhD degree on Computer Science from the University of Extremadura in 2002 and 2016 respectively. Formerly a software consultant, he is an associate professor at the Dept. of Computer Systems Engineering of the University of Extremadura (Spain). His research interests are in analytical performance models on heterogeneous platforms and the usage of deep and rein- forcement learning techniques to HPC platforms problems, including scheduling, process deployment and mapping, load balancing and communication modeling. Other research interest include current MPI implementations and applications. He codevelops AzequiaMPI, an efficient thread-based full MPI 1.3 standard implementation.

Escuela Politecnica
Av. de la Universidad, S/N, 10003
Caceres, Spain
Phone: 0034927257000. Ext. 51655
Email: jarico,juanmariohaut{@unex.es}

Dr. Juan Antonio Rico Gallego y Dr. Juan Mario Haut Hurtado como directores de la tesis titulada "Desarrollo de técnicas de aprendizaje automático para la optimización de aplicaciones científicas y de procesamiento masivo de datos en entornos de altas prestaciones", acreditan la colaboración y aportación del doctorando en el trabajo. Asímismo, se certifica también el factor de impacto junto con la respectiva categoría de la siguiente publicación incorporada en la tesis doctoral. Para cualquier aclaración con respecto a lo indicado, por favor, contacten con cualquiera de los directores.

Juan Antonio Rico Gallego PhD and Juan Mario Haut Hurtado PhD as directors of the PhD thesis titled "Development of machine learning techniques for the optimization of scientific and massive data processing applications in high performance computing environments", certify the collaboration and contribution of the doctoral student at work. Likewise, the impact factor is also certified along with the respective category of the following publication incorporated in the doctoral thesis. For any clarification regarding what is indicated, please contact any of the directors.

Artículo / Paper

Authors: M. E. Paoletti, X. Tao, J. M. Haut, **S. Moreno-Álvarez** and A. Plaza
Title: Deep mixed precision for hyperspectral image classification.
Journal: Journal of Supercomputing.
Other Information: vol 77, pp 9190–9201, 2020.
DOI: 10.1007/s11227-021-03638-2.
Impact factor 2020: 2.474. Q2
Abstract: Hyperspectral images (HSIs) record scenes at different wavelength channels, providing detailed spatial and spectral information. How to storage and process this high-dimensional data plays a vital role in many practical applications, where classification technologies have emerged as excellent processing tools. However, their high computational complexity and energy requirements bring some challenges. Adopting low-power consumption architectures and deep learning (DL) approaches has to provide acceptable computing capabilities without reducing accuracy demand. However, most DL architectures employ single-precision (FP32) to train models, and some big DL architectures will have a limitation on memory and computation resources. This can negatively affect the network learning process. This letter leads these challenges by using mixed precision into DL architectures for HSI classification to speed up the training process and reduce the memory consumption/access. Proposed models are evaluated on four widely used data sets. Also, low and high-power consumption devices are compared, considering NVIDIA Jetson Xavier and Titan RTX GPUs, to evaluate the proposal viability in on-board processing devices. Obtained results demonstrate the efficiency and effectiveness of these models within HSI classification task for both devices.
Source codes: `https://github.com/mhaut/CNN-MP-HSI`.
Contribución del doctorado: Planteamiento de la hipótesis, desarrollo práctico, análisis y discusión de los resultados, elaboración y escritura del manuscrito.

Firma / Signature
Nov / Nov, 2021

Juan Antonio Rico Gallego

Juan Mario Haut Hurtado

The Journal of Supercomputing (2021) 77:9190–9201
https://doi.org/10.1007/s11227-021-03638-2

Deep mixed precision for hyperspectral image classification

M. E. Paoletti[1] · X. Tao[3] · J. M. Haut[2] · S. Moreno-Álvarez[3] · A. Plaza[3]

Accepted: 18 January 2021 / Published online: 3 February 2021

Abstract

Hyperspectral images (HSIs) record scenes at different wavelength channels, providing detailed spatial and spectral information. How to storage and process this high-dimensional data plays a vital role in many practical applications, where classification technologies have emerged as excellent processing tools. However, their high computational complexity and energy requirements bring some challenges. Adopting low-power consumption architectures and deep learning (DL) approaches has to provide acceptable computing capabilities without reducing accuracy demand. However, most DL architectures employ single-precision (FP32) to train models, and some big DL architectures will have a limitation on memory and computation resources. This can negatively affect the network learning process. This letter leads these challenges by using mixed precision into DL architectures for HSI classification to speed up the training process and reduce the memory consumption/access. Proposed models are evaluated on four widely used data sets. Also, low and high-power consumption devices are compared, considering NVIDIA Jetson Xavier and Titan RTX GPUs, to evaluate the proposal viability in on-board processing devices. Obtained results demonstrate the efficiency and effectiveness of these models within HSI classification task for both devices. Source codes: https://github.com/mhaut/CNN-MP-HSI.

Keywords Hyperspectral image · Deeplearning · Mixed precision

✉ J. M. Haut
juanmariohaut@unex.es

M. E. Paoletti
mpaoletti@unex.es

A. Plaza
aplaza@unex.es

[1] Department of Computer Architecture, University of Malaga, Malaga, Spain

[2] Department of Communication and Control Systems, National Distance Education University, Madrid, Spain

[3] Department Technology of Computers and Communications, University of Extremadura, Badajoz, Spain

1 Introduction

As one of the most important data sources in remote sensing, hyperspectral imaging (HSI) provides rich and detailed spectral information about different materials on the earth surface. This spectral information has been widely used in a vast number of applications, such as image super-resolution [13] or spectral unmixing [24], among others. By analysing the continuous and narrow-band spectral signatures contained into pixels, different land cover categories can potentially be precisely differentiated. In this context, supervised classification plays a vital role in analysing and processing the HSI information. Moreover, the main interest of supervised classifiers lies in their guided training procedure, where labelled training samples are used to fit the model parameters. As a result, supervised methods achieve much higher precision values than unsupervised methods [6]. Therefore, many supervised methods have been extensively exploited in many practical activities, including land changes monitoring or natural resources management [1]. However, the accuracy of supervised classifiers is highly affected when the number of available training samples is limited in relation to the (high) data dimensionality, which is quite common in HSI due to the high cost and workload involved in the expert annotation. This may cause incomplete training process that easily introduces overfitting.

To address the aforementioned issues, several deep models have been developed reaching some breakthrough accomplishment in ML. Convolutional neural networks (CNNs) have demonstrated to be highly accurate techniques due to their ability for automatically extracting discriminative features. For instance, Yue et al. [26] proposed a CNN3D to classify HSI data in consideration of spectral-spatial information. To further enhance the accuracy, many improvements have been included to the CNN framework, such as Paoletti et al. [18] who presented a fast end-to-end CNN3D in the consideration of the full spectral signatures contained in HSI data to improve the classification accuracy. However, CNN models still encounter some limitations due to the depth and the high number of trainable parameters, and the intrinsic characteristics of HSI data. Specially, CNN models require an important amount of training data to adjust their weights appropriately. To overcome the limitations, different strategies have been proposed, such as semisupervised and active learning (AL) techniques [7], which address the lack of training samples problem by increasing the quantity-quality trade-off of the training samples, respectively. Particularly, these techniques are quite effective for discovering feature representativeness and discriminativeness. Other algorithms are based on residual and dense connections [20]. These techniques can alleviate both loss of information and vanishing gradient problems of very complex and deep architectures. Finally, there is a third type of algorithms based on new information routing techniques, such as capsule modules [19] which were proposed to overcome the CNN limitations when exploring the spatial relationships among learned instantiation parameters, such as size, perspective and orientation.

Despite these progresses, CNN-based models still face some challenges when dealing with HSI data. For instance, their training and inference stages require

processing thousands of millions of floating point operations, involving a large number of parameters and data variables which are in single-precision format. This increases the storage and compute requirements, which are already high because of the large spectral dimension. As a result, deep HSI classifiers require powerful and usually expensive computer resources, which are high power-consuming and therefore usually located in processing centres at the ground segment. As a result, developing on-board processing solutions is quite difficult, as computing platforms usually have limitations in both memory capacity and energy consumption [5]. In this context, adapting these deep models to reduce both memory and computing requirements is mandatory, not only to optimise computing times, but also to deploy them on low-power-consumption platforms, allowing on-board processing.

This paper adopts the newest trends in deep learning (DL), combining single (FP32) and half-precision (FP16) data and arithmetic formats to alleviate the high computational burden involved by deep HSI classification models while maintaining the accuracy performance [9, 17]. Indeed, this paper conducts an extensive comparative between several CNN-based models with single and mixed precision strategies by taking advantages of GPU architecture. Moreover, it provides a comparative between high and low-power consumption devices, comparing the NVIDIA GPUs Titan RTX and Jetson Xavier. Implemented HSI classifiers exhibit: (i) a high speed within math-intensive operations, where Tensor Cores optimize the convolution and matrix operations, and (ii) a significant reduction in both memory consumption and memory access times.

2 Background

The success of CNN lies in the application of local kernels onto small patches of the input, which filter the presence of specific features, followed by an activation function that obtains the neuronal response to those features:

$$\mathbf{Y} = \mathcal{H}(\mathbf{X} * \mathbf{W} + \mathbf{b}), \quad \text{with} \quad y_{i,j,t} = \sum_{\hat{i},\hat{j},\hat{t}} x_{i+\tilde{i},j+\tilde{j},\hat{t}} w_{\hat{i},\hat{j},\hat{t},t} + b_t \tag{1}$$

where $\mathbf{X} \in \mathbb{R}^{N \times N \times C}$ and $\mathbf{Y} \in \mathbb{R}^{M \times M \times K}$ are the input and output volumes, and $\mathbf{W} \in \mathbb{R}^{P \times P \times C \times K}$ and $\mathbf{b} \in \mathbb{R}^{K}$ defines the kernel weights and bias vector. Naturals $\hat{t} \in [1, C]$ and $t \in [1, K]$ index the input and output channels, $i, j \in [1, N]$ and $\hat{i}, \hat{j} \in [1, P]$ are indexing the output and kernel elements along the spatial dimension, and $\tilde{i}, \tilde{j}$ are the re-centred spatial indices indexing the input. $\mathcal{H}$ defines the activation function. From Eq. (1), it is easy to estimate the number of parameters and FLOPs performed by L convolution layers:

$$\text{Params.:} \quad \sum_{l=1}^{L} P_l^2 \times C_l \times K_l, \quad \text{FLOPs}_l : K_l \times N_l^2 \times C_l \times P_l^2 \tag{2}$$

This results in a massive amount of both trainable parameters and operations, making CNNs computational-intensive models. Important efforts have been conducted in order to reduce this drawback [4, 14]. Some approaches attempt to lighten the model by dividing the standard layer into separable depth/point-wise layers [23]. In [22], authors combine separable layers with shift-based operations to simulate the movement of the spatial features at each map, reducing the number of parameters and therefore the operations performed. Also, [2] proposes a lightweight model based on separable convolutions, combining the deep feature extraction with an approximate rank-order clustering. Furthermore, [21] divides the layer into a lighter convolution operation combined with a set of cheaper linear transformations. However, these models operate on FP32, without taking advantage of the benefits of mixed precision between FP32 and FP16.

3 Methodology

3.1 NVIDIA hardware optimization

The great demand for high performance devices for parallel and massive data processing has produced a wide range of GPUs, with different features and capabilities, which are pretty useful for many computationally intensive applications. In particular, the NVIDIA *Turing* architecture [10] is quite interesting, as it encodes an instruction and its control information with 128 bits, following the *Volta* encoding format, while previous architectures, such as *Pascal* and *Maxwell*, employ 64 bits for an instruction information and 64 bits of control. In this sense, *Turing* uses 91 bits for the instruction information and 23 bits for the control, while the remaining 14 are not used. Regarding memory, *Turing* combines L1 data cache and shared memory, following *Volta* model. The L2 cache is similar to previous generations, as it is unified for data, instructions, and memory. Table 1 provides a comparison between different NVIDIA GPUs architecture generations.

Therefore, *Turing* provides both a new streaming multiprocessor (SM) and a new memory system architectures, improving the efficiency and performance for massive parallel applications, which is quite interesting for optimizing DNNs for HSI classification. For instance, the *Turing* SM enables concurrent execution of FP32 and

Table 1 Comparative between several NVIDIA GPUs architectures

Architectures (sizes in KiB)					
Feature	*Turing*	*Volta*	*Pascal*	*Maxwell*	*Kepler*
2 cache	4096	6144	4096	2048	1536
L1.5 cache	46	64	64	32	–
L1 cache	144	256	32	32	48
L0 cache	16	12	12	–	–
Registers	64	64	128	64	255
Tensor Cores	✓	✓	✗	✗	✗

INT32 operations by including independent integer and floating-point math datapaths, while the different memory-level paths have been unified, providing a bigger and faster L2 cache and including support for the GDDR6 memory technology to increase the bandwidth. In addition, the architecture enhances the specialized execution units for matrix operations, called *Turing* Tensor Cores (TTCs). These TTCs are designed for the multiplication of two matrices, and the accumulation of a third matrix and provide multi-precision computing (considering FP32, FP16, INT8 and INT4). This property facilitates the combination of different precision formats (the so-called mixed precision) when training neural networks, which significantly accelerates arithmetic calculations and reduces the memory accesses/consumption, while keeping constant the accuracy results.

To properly optimize those operations, a high-performance CUDA SDK is provided, which includes the TensorRT as the inference optimizer. It takes a trained network (i.e. a network definition and the corresponding set of trained parameters) and produces a highly optimized runtime engine which performs the inference stage for the selected network. Finally, previous architectures can be built in embedded systems and other devices, such as the low-power-consumption and heterogeneous multi-processor NVIDIA Tegra models. For instance, the Tegra K1, TK1 and TK2 models use the *Kepler* architecture with a maximum of 365 GFLOPS and 951 MHz frequency, while Tegra X1 uses the *Maxwell* architecture. This model includes the half-precision, and it is composed of 256 GPU cores with a maximum of 1298 GFLOPS for FP16, 649 GFLOPS for FP32 and 1267 MHz frequency. Its successor, Tegra X2, changes the GPU architecture to *Pascal* with a maximum of 1465 GFLOPS for FP16, 750 GFLOPS for FP32 and 1267 MHz frequency. Tegra Xavier models adopt the *Volta* architecture with a maximum of 2820 GFLOPS for FP16, 1410 GFLOPS for FP32 and 1377 MHz frequency. Finally, although it is not yet publicly available to the market, Clara AGX models will implement the *Turing* architecture, considering the previous Xavier computing module to produce an optimal inference using TTCs.

3.2 Mixed precision strategy

As noted above, DL training systems use the FP32 format, which involves reading and computing millions of parameters represented by 32 bits. As a result, DNNs exhibit high time and energy costs. In this sense, the current trend is to reduce the memory footprint while optimizing the memory load of the model in order to speed the computation. This can be done by rearranging the data [25] or shrinking the data size [17]. In particular, casting some data/operations from FP32 to FP16 calculation is quite interesting, saving memory while increasing the speed up. Therefore, this paper studies the performance of several DNNs considering single and mixed precision [17]. Proposed implementations are based on TTC behaviour, where the DNNs performance is accelerated by transforming the vast majority of the mathematical computations into half-precision, while ensuring that no task-specific accuracy is degraded by identifying those steps/computations that require single precision. Hereof, hidden layers cast their FP32 inputs into FP16 format to conduct FP16

calculations, resulting into a FP16 outputs. Moreover, to prevent model accuracy loss, the following three techniques are implemented [9]: (i) maintaining a FP32 copy of weights, which accumulates the gradients after each optimizer step; (ii) loss-scaling to avoid vanishing gradient because data truncation, and (iii) FP16 arithmetic with accumulation in FP32. These techniques are detailed below.

During training step, data tensors (model weights, activations and gradients values) are cast into FP16 format, which implies that all arithmetic operations will be performed in FP16 too. This enables a faster operation, consuming less memory bandwidth. Equation (3) shows the castings for forward (3a) and backward (3b) steps formulation:

$$\forall l \in [1, \dots, L], \left(\mathbf{Z}^{(l)} = \mathbf{W}^{(l)} + \mathbf{b}^{(l)}\right)_{FP32 \rightarrow FP16} \tag{3a}$$

$$\forall l \in [1, \dots, L], \left(\mathbf{W}^{(l)}_{new} = \mathbf{W}^{(l)}_{old} - \mu \frac{\partial \mathcal{L}}{\partial \mathbf{W}^{(l)}}\right)_{FP32 \rightarrow FP16} \tag{3b}$$

where $\mathbf{W}^{(l)}$ denotes the corresponding weights of the l-th layer, μ is the learning rate and $\partial \mathcal{L}/\partial \mathbf{W}^{(l)}$ the weight derivation, considering $\mathcal{L}$ as the loss function and L the number of layers.

However, FP16 has some disadvantages which can produce an incorrect training. Indeed, FP16-data are composed of 16 bits, which are organized as follows: the first bit is the sign bit, which gives + 1 or − 1, then the following 5 bits are used to code an exponent between − 14 and 15, while the fraction part comprises the remaining 10 bits. Compared to FP32, it has a smaller range of possible values (from 2^{-24} to $(2 - 2^{-10})2^{15}$, where normalized value exponents range into $[-14, 15]$) but also a smaller offset. As a result smaller magnitudes than 2^{-24} are ignored and discarded. This produces severe precision errors, such as an incorrect weight update, zero gradients, or infinite/NaN values in the loss or activations.

To counteract this, first a FP32 copy of weights is kept. Although the FP16-weights are used during the forward and backward steps of each iteration to halve the memory consumption and bandwidth, weights updating is conducted over the FP32-weights to avoid the truncated/zero weights. Then, updated weights are cast back to FP16 to perform the next forward–backward steps. Regarding the gradients, the loss is scaled up by factor $s = 8$ before the backpropagation, pushing it into large values to prevent potentially truncated/zero gradients, and then a reverse scaling is applied before the weight update. In this sense, the softmax of the model tail should be represented in FP32 to avoid numerical error calculations. Equation (4) shows the loss scaling formulation for the weight update:

$$\mathbf{W}_{FP32} = \mathbf{W}_{FP32} - \mu \frac{1}{s}\left(s \frac{\partial \mathcal{L}}{\partial \mathbf{W}}_{FP16 \rightarrow FP32}\right) \tag{4}$$

This simple and cheap solution allows the computation of small gradients, avoiding artificial vanishing problems and poor weights to be learnt, and hence, improving the convergence of the model as demonstrated in [16, 17]. Finally, DNN can include some operations that required FP32 precision to avoid the uncertainties (in particular batch-normalization and softmax layers). Indeed, DNN performs vector

dot-products, reductions and point-wise operations, where the FP16 vector dot-product has to accumulate the partial products into a FP32 value and then convert it to FP16 before writing to memory in order to maintain the accuracy precision. This can be easily conducted by the TTCs. Moreover, large reductions which perform the sum of a vector elements should be done in FP32 following the same procedure. Figure 1 provides the mixed precision training flowchart.

As we can observe, the DNN has to detect the operation types that should be represented in FP32 because their FP16 representation does not provide the desired accuracy (for instance, the loss, softmax and cross-entropy functions). To do this, Apex Automatic Mixed Precision (AMP) [11] is used. Apex is an extension on PyTorch which takes care of tensor casting, ensuring that all arguments are of the same type. To ensure that castings are performed only once per iteration, it keeps an internal cache of all the casts of the parameters and reuses them. Moreover, it performs the FP32 copy of weights, computing forward and backward stages in FP16 and updating in FP32, and detecting those operations that should be executed on FP32 while most of computations are executed on FP16.

4 Experimental evaluation

To conduct the comparison between several DNNs for HSI classification in single and mixed precision formats, an X Generation Intel Core i9-9940X is employed, which is composed of 14 cores (28 with multi-task processing) running at 4.40GHz with a 19M cache. The GPU is an NVIDIA Titan RTX with 24GB of GDDR6 memory, 4608 cores and 576 *Turing* Tensor Cores. Also, the NVIDIA

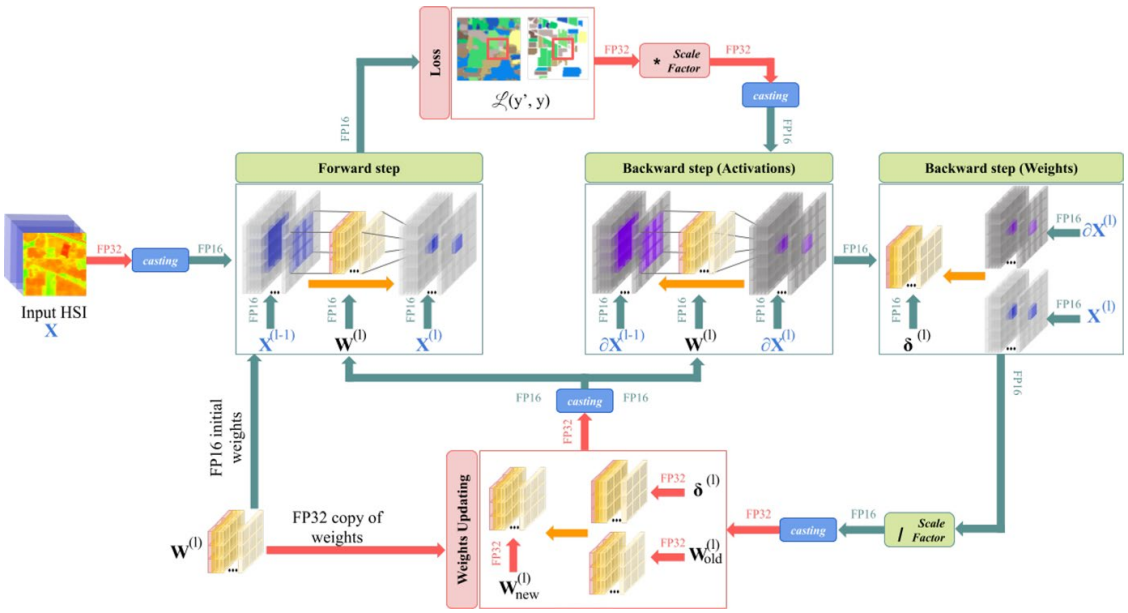


Fig. 1 Mixed precision training. Red and green paths represent FP32 and FP16 formats, respectively. During forward step, $\mathbf{X}^{(l-1)}$ and $\mathbf{X}^{(l)}$ denote the input and output activations of l-th layer, and $\mathbf{W}^{(l)}$ its kernel-weights. During backpropagation, $\partial\mathbf{X}^{(l)}$ and $\delta^{(l)}$ are the activation and weights gradients

Jetson AGX Xavier is tested, with an ARM CPU composed of 8 cores running at 2.5GHz with 16GB-RAM and 512 *Volta* tensor cores.

Moreover, four widely used HSI data sets [3, 12] have been used in our experiments, i.e. Indian Pines (IP), Salinas Valley (SV), Kennedy Space Center (KSC) and the University of Pavia (UP).

- *IP* has 145×145 pixels with 20m spatial resolution and 224 spectral bands in the wavelength range from 0.4 to 2.5 µm. We remove 24 bands due to water absorption and null values, keeping 200 bands.
- *UP* has 610×340 pixels and 103 spectral bands in the wavelength range from 0.43 to 0.86 µm and 1.3 m spatial resolution.
- *SV* has 224 spectral bands and 512×217 pixels with a 3.7 m spatial resolution in the range from 0.4 to 2.5 µm. We remove some noisy and water absorption bands from the original 224 bands, keeping 200 bands.
- *KSC* has 224 spectral bands and 512×614 pixels with a 18 m spatial resolution in the range from 0.4 to 2.5 µm. Again, we remove water absorption and low SNR bands, keeping 176 bands.

Overall (OA) and average accuracy (AA) and kappa coefficient (K) have been used to measure the accuracy performance. Also, the runtime (in seconds), number of parameters, memory consumption (in MB) and the processed-samples-per-second (PSpS) measurements have been adopted to evaluate the computational properties.

We compare the performance of several DNNs for HSI classification, considering single and mixed precision training and inference. To address the experimentation, we have selected the most innovative algorithms. Thus, effective algorithms previously tested on a broad set of algorithms from the HSI DL literature are used. In particular, the CNN2D and CNN2D+RO [8] have been used to perform spatial feature extraction (FE) and HSI classification, where the second one includes random occlusion data augmentation. Regarding the scalability and consistency of our proposal, the deeper and more complex PResNet model [20] is used to conduct deep spectral-spatial FE and HSI classification. Model setting specifications [8, 20] have been followed. These algorithms were previously tested over the same data sets mentioned above, maintaining a consistency in the evaluation conducted.

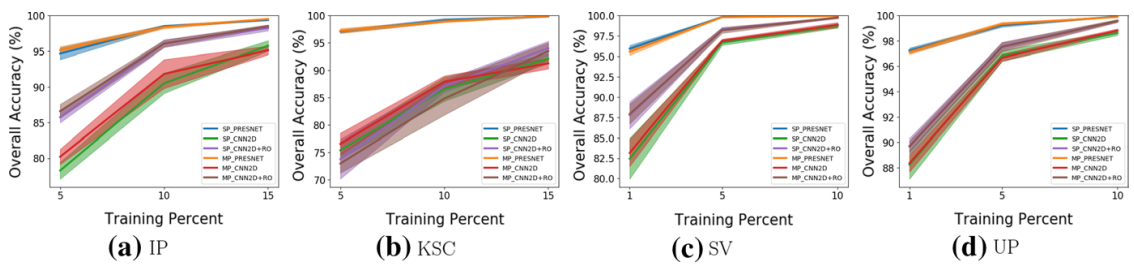


Fig. 2 Overall accuracy (%) of SP_CNN2D, SP_CNN2D+RO and SP_PResNet and their mixed precision counterparts, MP_CNN2D, MP_CNN2D+RO and MP_PResNet, when considering different training percentages in IP, KSC, SV and UP scenes

Figure 2 provides the OA evolution of CNN2D, CNN2D-RO and PResNet in single (SP_CNN2D, SP_CNN2D+RO and SP_PResNet) and automatic mixed precision (MP_CNN2D, MP_CNN2D+RO and MP_PResNet) when considering different amounts of training samples, in particular 5%, 10% and 15% of IP and KSC scenes, and 1%, 5% and 10% of SV and UP images. Focusing on FP32 implementations, SP_PResNet significantly outperforms the OA of CNN models (with the exception of KSC, due to the spatial complexity and intense spectral mixing of the image), while the SP_CNN2D+RO consequently improves the results of the conventional CNN2D, which is greatly affected by overfitting. This behaviour can be observed also between MP implementations, where MP_PResNet is able to extract highly discriminative feature representations, as it not only builds a more complex architecture but also considers the full spectral information. Comparing SP with MP implementations, each pair of networks reach close accuracy with slight variations, in particular SP_CNN2D and MP_CNN2D, where the second one usually reaches better OA, i.e. with MP, the problem of overfitting can be effectively reduced.

Table 2 provides a deeper study between both implementations, evaluating the accuracy in terms of OA, AA and Kappa values over IP (10% training data) and UP (5% training data). Once more, PResNet reaches the best accuracy values, and it is followed by CNN2D+RO model. Also, the pairs of models achieve quite similar results, being sometimes slightly better the SP-version and others the MP (in particular the CNN2D). Focusing on the number of parameters, pairs of implementations exhibit also quite similar parameters, where the PResNet comprises more parameters than the other models. However, runtimes captured during training stage show that, despite having the same number of parameters (approximately), the MP-based implementation is faster than SP models, achieving also similar accuracy performances.

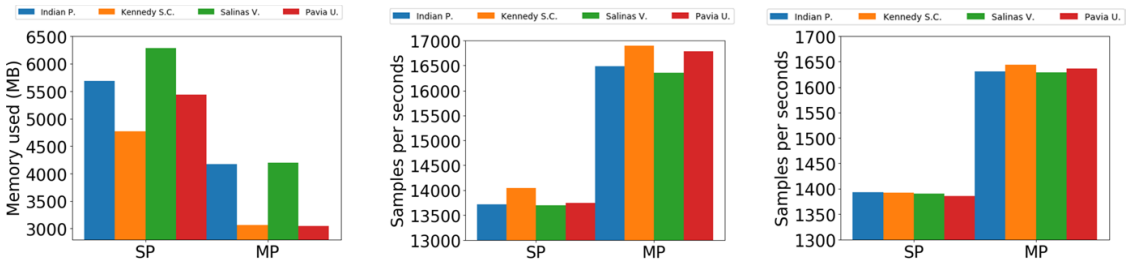
Finally, Fig. 3 provides the memory consumption and the PSpS ratios of both PResNet implementations considering every HSI dataset. Moreover, we compare both GPUs, evaluating the performance of typically high-performance computing and low-power-consumption devices. Regarding the Titan RTX, and focusing on the memory requirements, the same batch size [20] is used, where SV and IP scenes consume the most due their number of channels. However, MP_PResNet greatly reduces its memory consumption, where the FP32 copy of weights and the auxiliar FP32 variables barely represent a great effort compared to the large number of activations, which are stored in FP16. Moreover, the transformation of most operations to FP16 arithmetic significantly increases processing speed, while the reading/writing of FP16 tensors in memory has less impact. As a result, more samples can be processed per second, as we can observe in Fig. 3b, which in turn allows the batch size to be increased. This behaviour can be also observed on Jetson Xavier device, where the MP-implementations process more samples per second. Comparing both devices, we can see that the processing ratios are quite similar, where the Titan RTX has a higher ratio because of its optimized Turing Tensor Cores, while the Jetson Xavier contains Volta Tensor Cores. This result demonstrates that mixed-precision is an useful technique to reduce the high computational and memory requirements of DNNs for HSI classification, effectively adjusting them to the hardware limitations imposed by potential on-board platforms, such as the Jetson Xavier.

Table 2 Accuracy performance over IP and UP scenes considering single and mixed precision

Class	Indian Pines						Pavia University					
	PResNet		CNN2D		CNN2D+RO		PResNet		CNN2D		CNN2D+RO	
	SP	MP	SP	MP	SP	MP	SP	MP	SP	MP	SP	MP
0	97.07	99.02	73.66	69.75	90.24	76.1	99.8	99.76	95.94	95.83	97.61	97.23
1	99.41	99.27	86.27	86.52	92.09	92.48	99.99	99.98	98.66	98.85	99.37	99.22
2	98.74	98.8	84.26	89.64	95.34	95.82	99.05	98.99	88.76	90.49	91.42	91.68
3	96.71	94.74	95.87	93.52	95.02	95.87	99.6	99.46	96.63	96.87	98.91	98.64
4	99.59	98.94	89.06	87.68	95.35	93.98	100.0	100.0	99.48	99.84	98.79	97.89
5	99.54	99.36	93.94	95.37	97.9	98.78	100.0	99.98	92.98	94.15	97.9	98.18
6	88.8	89.6	80.8	88.0	96.0	90.4	98.46	98.48	91.72	90.06	94.43	95.77
7	99.91	100.0	95.58	97.02	97.4	97.91	99.76	99.71	97.68	97.22	98.97	98.67
8	96.66	95.55	54.44	76.67	81.11	73.33	99.89	99.85	97.95	98.09	98.73	99.07
9	96.66	97.01	89.19	92.41	94.56	93.35	–	–	–	–	–	–
10	98.92	98.72	91.96	93.67	97.76	97.89	–	–	–	–	–	–
11	97.12	97.11	85.28	83.9	94.68	93.15	–	–	–	–	–	–
12	99.89	100.0	95.03	91.24	97.08	98.05	–	–	–	–	–	–
13	98.54	98.7	97.12	98.65	98.89	99.51	–	–	–	–	–	–
14	95.33	94.41	90.37	90.66	95.27	96.71	–	–	–	–	–	–
15	95.24	95.95	81.19	79.52	96.91	93.1	–	–	–	–	–	–
OA	**98.49**	98.38	90.51	**91.82**	**96.06**	96.05	**99.82**	99.79	96.65	**96.87**	**98.28**	98.18
AA	**97.38**	97.32	86.50	**88.39**	**94.72**	92.90	**99.62**	99.58	95.53	**95.71**	97.35	**97.37**
K(x100)	**98.28**	98.15	89.17	**90.66**	**95.51**	95.49	**99.76**	99.72	95.55	**95.85**	**97.72**	97.58
Tr. time (s)	265.27	**256.6**	224.77	**218.57**	233.72	**222.69**	674.73	**600.1**	385.04	**331.69**	397.6	**341.12**
Parameters	4.6M		1.1M				4.6M		1.1M			

Also, number of parameters and training runtimes are provided

Best results presented in bold font



(a) Titan mem. use **(b)** Titan PSpS ratio **(c)** Xavier PSpS ratio

Fig. 3 Memory consumption and processed samples per second ratio on Titan RTX and Jetson Xavier devices considering PResNet

5 Conclusions and future work

Most existing DL architectures use FP32 format for training, brighting a big burden for memory consumption/access. To address the issue, we exploit mixed precision over two GPU devices (Titan RTX and Jetson Xavier) to train several deep HSI classifiers. This can reduce memory consumption and the time spent on memory and arithmetic operations. Experimental results have revealed that MP-based implementations are quite effective and efficient in HSI classification. Moreover, the comparison between high- and low-power-consumption devices shows that it is an useful solution to reduce the computational and memory requirements, adapting the deep and complex models to the limitations imposed by the potential on-board platforms. This provides interesting results for HSI classification in this type of devices where memory usage and workload computation are key factors, as demonstrated in recent studies on satellites devices [15]. Encouraged by these results, we will explore more DL models with mixed precision, exploring practical applications.

Acknowledgements Supported by FEDER and Junta de Extremadura (GR18060).

References

1. Bioucas-Dias JM, Plaza A, Camps-Valls G, Scheunders P, Nasrabadi N, Chanussot J (2013) Hyperspectral remote sensing data analysis and future challenges. IEEE Geosci Remote Sens Mag 1(2):6–36
2. Fang B, Li Y, Zhang H, Chan JCW (2020) Collaborative learning of lightweight convolutional neural network and deep clustering for hyperspectral image semi-supervised classification with limited training samples. ISPRS J Photogram Remote Sens 161:164–178. https://doi.org/10.1016/j.isprsjprs.2020.01.015
3. Green RO, Eastwood ML, Sarture CM, Chrien TG, Aronsson M, Chippendale BJ, Faust JA, Pavri BE, Chovit CJ, Solis M et al (1998) Imaging spectroscopy and the airborne visible/infrared imaging spectrometer (aviris). Remote Sens Environ 65(3):227–248
4. Haut JM, Alcolea A, Paoletti ME, Plaza J, Resano J, Plaza A (2020) Gpu-friendly neural networks for remote sensing scene classification. IEEE Geosci Remote Sens Lett. https://doi.org/10.1109/LGRS.2020.3019378
5. Haut JM, Bernabé S, Paoletti ME, Fernandez-Beltran R, Plaza A, Plaza J (2018) Low-high-power consumption architectures for deep-learning models applied to hyperspectral image classification. IEEE Geosci Remote Sens Lett 16(5):776–780

6. Haut JM, Paoletti M, Plaza J, Plaza A (2017) Cloud implementation of the k-means algorithm for hyperspectral image analysis. J Supercomput 73(1):514–529
7. Haut JM, Paoletti ME, Plaza J, Li J, Plaza A (2018) Active learning with convolutional neural networks for hyperspectral image classification using a new bayesian approach. IEEE Trans Geosci Remote Sens 56(11):6440–6461
8. Haut JM, Paoletti ME, Plaza J, Plaza A, Li J (2019) Hyperspectral image classification using random occlusion data augmentation. IEEE Geosci Remote Sens Lett 16(11):1751–1755
9. Jia X, Song S, He W, Wang Y, Rong H, Zhou F, Xie L, Guo Z, Yang Y, Yu L et al (2018) Highly scalable deep learning training system with mixed-precision: training imagenet in four minutes. arXiv preprint arXiv:1807.11205
10. Jia Z, Maggioni M, Smith J, Scarpazza DP (2019) Dissecting the nvidia turing t4 gpu via microbenchmarking. arXiv preprint arXiv:1903.07486
11. Kim D, Kwon Y, Liu P, Kim IL, Perry DM, Zhang X, Rodriguez-Rivera G (2016) Apex: automatic programming assignment error explanation. ACM SIGPLAN Notices 51(10):311–327
12. Kunkel B, Blechinger F, Lutz R, Doerffer R, Van der Piepen H, Schroder M (1988) Rosis (reflective optics system imaging spectrometer)-a candidate instrument for polar platform missions. In: Optoelectronic technologies for remote sensing from space, vol 868. International Society for Optics and Photonics, pp 134–141
13. Lanaras C, Baltsavias E, Schindler K (2015) Hyperspectral super-resolution by coupled spectral unmixing. In: Proceedings of the IEEE International Conference on Computer Vision, pp 3586–3594
14. Li P, Han L, Tao X, Zhang X, Grecos C, Plaza A, Ren P (2020) Hashing nets for hashing: a quantized deep learning to hash framework for remote sensing image retrieval. IEEE Trans Geosci Remote Sens 58(10):7331–7345. https://doi.org/10.1109/TGRS.2020.2981997
15. Lofqvist M, Cano J (2020) Accelerating deep learning applications in space. In: The 34th Annual Small Satellite Conference
16. Lu J, Lu S, Wang Z, Fang C, Lin J, Wang Z, Du L (2019) Training deep neural networks using posit number system. In: 32nd IEEE International System-on-Chip Conference (SOCC), pp 62–67. https://doi.org/10.1109/SOCC46988.2019.1570558530
17. Micikevicius P, Narang S, Alben J, Diamos G, Elsen E, Garcia D, Ginsburg B, Houston M, Kuchaiev O, Venkatesh G, Wu H (2018) Mixed precision training. In: International Conference on Learning Representations
18. Paoletti M, Haut J, Plaza J, Plaza A (2018) A new deep convolutional neural network for fast hyperspectral image classification. ISPRS J Photogramm Remote Sens 145:120–147
19. Paoletti ME, Haut JM, Fernandez-Beltran R, Plaza J, Plaza A, Li J, Pla F (2018) Capsule networks for hyperspectral image classification. IEEE Trans Geosci Remote Sens 57(4):2145–2160
20. Paoletti ME, Haut JM, Fernandez-Beltran R, Plaza J, Plaza AJ, Pla F (2018) Deep pyramidal residual networks for spectral-spatial hyperspectral image classification. IEEE Trans Geosci Remote Sens 57(2):740–754
21. Paoletti ME, Haut JM, Sidonio N, Plaza J, Plaza A (2021) Ghostnet for hyperspectral image classification. IEEE Trans Geosci Remote Sens. https://doi.org/10.1109/TGRS.2021.3050257
22. Paoletti ME, Haut JM, Tao X, Plaza J, Plaza A (2020) Flop-reduction through memory allocations within cnn for hyperspectral image classification. IEEE Trans Geosci Remote Sens. https://doi.org/10.1109/TGRS.2020.3024730
23. Roy SK, Chatterjee S, Bhattacharyya S, Chaudhuri BB, Platoš J (2020) Lightweight spectral-spatial squeeze-and- excitation residual bag-of-features learning for hyperspectral classification. IEEE Trans Geosci Remote Sens 58(8):5277–5290. https://doi.org/10.1109/TGRS.2019.2961681
24. Tao X, Cui T, Ren P (2019) Cofactor-based efficient endmember extraction for green algae area estimation. IEEE Geosci Remote Sens Lett 16(6):849–853
25. Yu J, Huang T (2019) Autoslim: towards one-shot architecture search for channel numbers. arXiv preprint arXiv:1903.11728
26. Yue J, Zhao W, Mao S, Liu H (2015) Spectral-spatial classification of hyperspectral images using deep convolutional neural networks. Remote Sens Lett 6(6):468–477

Escuela Politecnica
Av. de la Universidad, S/N, 10003
Caceres, Spain
Phone: 0034927257000. Ext. 51655
Email: jarico,juanmariohaut{@unex.es}

Dr. Juan Antonio Rico Gallego y Dr. Juan Mario Haut Hurtado como directores de la tesis titulada "Desarrollo de técnicas de aprendizaje automático para la optimización de aplicaciones científicas y de procesamiento masivo de datos en entornos de altas prestaciones", acreditan la colaboración y aportación del doctorando en el trabajo. Asímismo, se certifica también el factor de impacto junto con la respectiva categoría de la siguiente publicación incorporada en la tesis doctoral. Para cualquier aclaración con respecto a lo indicado, por favor, contacten con cualquiera de los directores.

Juan Antonio Rico Gallego PhD and Juan Mario Haut Hurtado PhD as directors of the PhD thesis titled "Development of machine learning techniques for the optimization of scientific and massive data processing applications in high performance computing environments", certify the collaboration and contribution of the doctoral student at work. Likewise, the impact factor is also certified along with the respective category of the following publication incorporated in the doctoral thesis. For any clarification regarding what is indicated, please contact any of the directors.

Artículo / Paper

Authors: J. M. Haut, M. E. Paoletti, **S. Moreno-Álvarez**, J. Plaza, J. A. Rico-Gallego and A. Plaza
Title: Distributed Deep Learning for Remote Sensing Data Interpretation.
Journal: Proceedings of the IEEE.
Other Information: vol 109.8, pp 1320-1349, 2021.
DOI: 10.1109/JPROC.2021.3063258.
Impact factor 2020: 10.961. Q1
Abstract: As a newly emerging technology, deep learning (DL) is a very promising field in big data applications. Remote sensing often involves huge data volumes obtained daily by numerous in-orbit satellites. This makes it a perfect target area for data-driven applications. Nowadays, technological advances in terms of software and hardware have a noticeable impact on Earth observation applications, more specifically in remote sensing techniques and procedures, allowing for the acquisition of data sets with greater quality at higher acquisition ratios. This results in the collection of huge amounts of remotely sensed data, characterized by their large spatial resolution (in terms of the number of pixels per scene), and very high spectral dimensionality, with hundreds or even thousands of spectral bands. As a result, remote sensing instruments on spaceborne and airborne platforms are now generating data cubes with extremely high dimensionality, imposing several restrictions in terms of both processing runtimes and storage capacity. In this article, we provide a comprehensive review of the state of the art in DL for remote sensing data interpretation, analyzing the strengths and weaknesses of the most widely used techniques in the literature, as well as an exhaustive description of their parallel and distributed implementations (with a particular focus on those conducted using cloud computing systems). We also provide quantitative results, offering an assessment of a DL technique in a specific case study (source code available: `https://github.com/mhaut/cloud-dnn-HSI`). This article concludes with some remarks and hints about future challenges in the application of DL techniques to distributed remote sensing data interpretation problems. We emphasize the role of the cloud in providing a powerful architecture that is now able to manage vast amounts of remotely sensed data due to its implementation simplicity, low cost, and high efficiency compared to other parallel and distributed architectures, such as grid computing or dedicated clusters.
Contribución del doctorado: Planteamiento de la hipótesis, desarrollo práctico, análisis y discusión de los resultados, elaboración y escritura del manuscrito.

Firma / Signature
Nov / Nov, 2021

Juan Antonio Rico Gallego

Juan Mario Haut Hurtado

INVITED PAPER

Distributed Deep Learning for Remote Sensing Data Interpretation

A comprehensive review of the state-of-the-art in deep learning for remote sensing data interpretation is given. The pros and cons of the most widely used techniques in the literature are analyzed, as well as their parallel and distributed implementations. The article concludes with some remarks about future challenges in the application of deep learning techniques to distributed remote sensing data interpretation problems.

By Juan M. Haut, *Senior Member IEEE*, Mercedes E. Paoletti, *Senior Member IEEE*, Sergio Moreno-Álvarez, Javier Plaza, *Senior Member IEEE*, Juan-Antonio Rico-Gallego, and Antonio Plaza, *Fellow IEEE*

ABSTRACT | As a newly emerging technology, deep learning (DL) is a very promising field in big data applications. Remote sensing often involves huge data volumes obtained daily by numerous in-orbit satellites. This makes it a perfect target area for data-driven applications. Nowadays, technological advances in terms of software and hardware have a noticeable impact on Earth observation applications, more specifically in remote sensing techniques and procedures, allowing for the acquisition of data sets with greater quality at higher acquisition ratios. This results in the collection of huge amounts of remotely sensed data, characterized by their large spatial resolution (in terms of the number of pixels per scene), and very high spectral dimensionality, with hundreds or even thousands of spectral bands. As a result, remote sensing instruments on spaceborne and airborne platforms are now generating data cubes with extremely high dimensionality, imposing several restrictions in terms of both processing runtimes and storage capacity. In this article, we provide a comprehensive review of the state of the art in DL for remote sensing data interpretation, analyzing the strengths and weaknesses of the most widely used techniques in the literature, as well as an exhaustive description of their parallel and distributed implementations (with a particular focus on those conducted using cloud computing systems). We also provide quantitative results, offering an assessment of a DL technique in a specific case study (source code available: https://github.com/mhaut/cloud-dnn-HSI). This article concludes with some remarks and hints about future challenges in the application of DL techniques to distributed remote sensing data interpretation problems. We emphasize the role of the cloud in providing a powerful architecture that is now able to manage vast amounts of remotely sensed data due to its implementation simplicity, low cost, and high efficiency compared to other parallel and distributed architectures, such as grid computing or dedicated clusters.

KEYWORDS | Big data; cloud computing; deep learning (DL); parallel and distributed architectures; remote sensing.

NOMENCLATURE

EO	Earth observation.
GPUs	Graphics processing units.
CPUs	Central processing units.
AVIRIS	Airbone Visible/Infrared Imaging Spectrometer.

Manuscript received January 22, 2021; revised February 23, 2021; accepted February 28, 2021. Date of publication March 15, 2021; date of current version July 19, 2021. This work was supported in part by the Junta de Extremadura under Grant GR18060, in part by the Spanish Ministerio de Ciencia e Innovación under Project PID2019-110315RB-I00 (APRISA), in part by the European Union's Horizon 2020 Research and Innovation Program under Grant Agreement 734541 (EOXPOSURE), and in part by the Computing Facilities of Extremadura Research Centre for Advanced Technologies (CETA-CIEMAT) funded by the European Regional Development Fund (ERDF); CETA-CIEMAT belongs to CIEMAT and the Government of Spain. *(Corresponding author: Antonio Plaza.)*

Juan M. Haut is with the Department of Communication and Control Systems, Higher School of Computer Engineering, National Distance Education University, 28015 Madrid, Spain (e-mail: juanmariohaut@unex.es).

Mercedes E. Paoletti is with the Department of Computer Architecture, School of Computer Science and Engineering, University of Málaga, 29071 Málaga, Spain (e-mail: mpaoletti@unex.es).

Sergio Moreno-Álvarez and **Juan-Antonio Rico-Gallego** are with the Department of Computer Systems Engineering and Telematics, University of Extremadura, 10003 Cáceres, Spain (e-mail: smoreno@unex.es; jarico@unex.es).

Javier Plaza and **Antonio Plaza** are with the Hyperspectral Computing Laboratory, Department of Technology of Computers and Communications, Escuela Politécnica, University of Extremadura, 10003 Cáceres, Spain (e-mail: mpaoletti@unex.es; jplaza@unex.es; aplaza@unex.es).

Digital Object Identifier 10.1109/JPROC.2021.3063258

AVIRIS-NG	Airbone Visible/Infrared Imaging Spectrometer New Generation.
NASA	National Aeronautics and Space Administration.
TB	Terabyte.
GB	Gigabyte.
PB	Petabyte.
mpp	Meters per pixel.
EnMAP	Environmental Mapping and Analysis Program.
EOSDIS	Earth Observing System Data and Information System.
NSMC	China National Satellite Meteorological Center.
CCRSDA	China Center for Resources Satellite Data and Application.
ESA	European Space Agency.
GEO	Group on Earth Observations.
HPC	High-performance computing.
SAR	Synthetic aperture radar.
LIDAR	Light Detection and Ranging.
OLI	Operational land imager.
ASTER	Advanced Spaceborne Thermal Emission and Reflection Radiometer.
VNIR	Visible and near-infrared.
TIR	Thermal infrared.
SWIR	Shortwave infrared.
GSD	Ground sample distance.
PAN	Panchromatic.
MSIs	Multispectral images.
HSI	Hyper-spectral images.
PCA	Principal component analysis.
TCT	Tasseled cap transformation.
WT	Wavelet transform.
KNN	K-nearest neighbor.
DTs	Decision trees.
RFs	Random forests.
MLR	Multinomial logistic regression.
GMMs	Gaussian mixture models.
NB	Naive Bayes-based approaches.
HMMs	Hidden Markov models.
SVMs	Support vector machines.
ANNs	Artificial neural networks.
SAEs	Stacked autoencoders.
DBNs	Deep belief networks.
RNNs	Recurrent neural networks.
CNNs	Convolutional neural networks.
DNNs	Deep neural networks.
RBMs	Restricted Boltzmann machines.
LSTM	Long short-term memory.
RESFlow	Remote Sensing data Flow.
InSPIRE	Integrated Sustainable Pan-European Infrastructure for Researchers in Europe.
XSEDE	Extreme Science and Engineering Discovery Environment.
MTFC	Multi-GPU training framework.
PNPE	Parallel neighbor pixel extractor.
G-POD	Grid processing on demand.
GENESI-DR	Ground European Network for Earth Science Interoperations and Digital Repositories.
GiSHEO	Grid Services for training and High Education in Earth Observation.
CEOS	Committee on Earth Observation Satellites.
DGGS	Discrete global grid systems.
CBIR	Content-based image retrieval.
ICP	InterIMAGE cloud Platform.
HDFS	Hadoop file system.
YARN	Yet Another Resource Negotiator.
RDDs	Resilient distributed data sets.
VMs	Virtual machine.
SaaS	Software as a Service.
PaaS	Platform as a Service.
IaaS	Infrastructure as a Service.
API	Application programming interface.
SDNs	Software-defined networks.
DNs	Domain name servers.
GFS	Google File System.
DAG	Directed acyclic graph.
I/O	Input–output.
HPCC	High-performance computing challenger.
AWS	Amazon Web Services.
HPL	High-performance linpack.
EC2	Elastic computing cloud.
GCE	Google Compute Engine.
IBM SL	IBM SoftLayer.
MRAP	MapReduce with Access Patterns.
MPI	Message passing interface.
BLAS	Basic linear algebra subprogram.
ML	Machine learning.
DL	Deep learning.
OS	Operating system.
MAP	Maximum *a posteriori*.
BS	Block size.
MLP	Multilayer perceptron.
ReLU	Rectified linear unit.
BFGS	Broyden–Fletcher–Goldfarb–Shanno.
DGEMM	Double-precision matrix-matrix multiplication.
TF	TensorFlow.
PS	Parameter server.
IP	Indian Pines.
BIP	Big Indian Pines.
HDD	Hard disk drive.
RAM	Random access memory.
FLOPs	Floating-point operations.
PRACE	Partnership for Advanced Computing in Europe.
UCI	Unified cloud interface.

I. INTRODUCTION

A. Big Remote Sensing Data

Remotely sensed images provide very detailed information about the surface of the Earth, which often results

in very significant computational requirements. Present and future missions for EO have significantly increased the spatial, spectral, and temporal resolutions of existing imaging instruments, resulting in higher data volumes. Meanwhile, the variety of multisensor and multiresolution acquisitions inevitably leads to the gradual acceptance of the generated data sets as "big remote sensing data" [1], not merely due to their high volume but also due to the inherent complexity of the data, which calls for advanced processing techniques that often use external sources of information (e.g., social media data) for adequate interpretation [2].

For instance, remotely sensed hyperspectral images [3] record information using hundreds of spectral bands collected at nearly contiguous wavelengths in the electromagnetic spectrum. This significantly increases their volume with regards to other kinds of image data used in remote sensing applications, such as MSIs (tens of bands), radar, or microwave data sets, creating important requirements in terms of storage and processing. In fact, there has been an exponential growth in these requirements due to recent technological advances in both the quality and number of available imaging instruments. This results from the ever-increasing number of EO missions that are now generating a nearly continuous flow of remotely sensed data, which fostered the creation of large remote sensing data repositories that can only be exploited using adequate parallel and distributed processing techniques [4]. Only in the hyperspectral domain, the AVIRIS [5] and its new generation (AVIRIS-NG)—operated by NASA's Jet Propulsion Laboratory—acquires almost 9 GB of data per hour. Similarly, the EO-1 Hyperion sensor acquires about 71.9 GBs of data per hour, which means over 1.6 TB of data per day. Most of the EO missions that will be operational soon, e.g., the German EnMAP,[1] exhibit equal or even higher data acquisition rates. This confirms that remote sensing data have entered the "big data era" [6]. To be more specific, we summarize, in the following, the properties that qualify remote sensing data as a kind of big data [7].

1) *Data Volume:* As mentioned before, remote sensing data in the optical, radar, and microwave domains are now characterized by their huge dimensionality, which results in the acquisition of several TBs of data per day. The total amount of data archived by the EOSDIS[2] now exceeds 30 PB of data. The amount of archived data at the NSMC[3] exceeds 5 PBs, and the CCRSDA[4] has more than 20 PBs of remote sensing data in its archive.
2) *Data Variety:* According to the state of the satellite industry report,[5] there are more than 300 EO satellites currently in orbit. All of them carry at least one EO instrument, and they are able to collect and transmit data continuously to Earth stations. This means that hundreds of different kinds of remotely sensed data are transmitted (in parallel) to the respective ground receiving stations every day. In addition, since Landsat-1 was first operational in 1972, there have been more than 500 EO satellites launched into space, collecting and archiving more than 1000 different types of remote sensing data.
3) *Data Velocity:* With the development of satellite constellations, the satellite revisit times have gradually transitioned from months to days, hours, or even minutes. As a result, the multitemporal resolution of remote sensing data has increased exponentially, allowing for advanced environmental monitoring and climate change applications. As mentioned before, data centers now receive an almost continuous flow of remote sensing data at ever-increasing speeds, which creates important requirements in terms of storage and analysis (particularly in the context of real-time applications).

In addition to the three aforementioned characteristics, commonly known as the "three V's" of remote sensing big data [6], there are also other important aspects inherent to the analysis and interpretation of such data. One of the most important ones is data *heterogeneity*. Specifically, due to the existence of various satellite orbits and specifications for different sensors (with different storage formats, data projections, spatial resolutions, and revisit times), there are vast differences in the formats of the archived data, and these differences create difficulties when developing general data interpretation techniques. Currently, big remote sensing data analysis is attracting significant attention from governmental and commercial partners, as well as from academic institutions. This is because the high-level products that can be obtained from the data are useful in many relevant applications, including agriculture, disaster prevention and reduction, environmental monitoring, public safety, and urban planning, among many others [8].

One of the most important remote sensing big data projects has been the EOSDIS [9], which provides end-to-end capabilities for managing NASA's Earth science data from various sources. In Europe, the ESA has been organizing the "Big Data from Space" conference, with the ultimate goal of stimulating interactions and bringing together partners and service providers willing to exploit and interpret remotely sensed big data collected from space. The GEO [10], a large-scale cooperation organization, also promotes the development of big data in remote sensing applications. As for commercial applications, Google Earth[6] is a successful case of bringing remotely sensed big data to a large number of users around the world. Many remote sensing applications, such as target detection, land-cover classification, spectral unmixing, and pansharpening, can now be developed easily by resorting to Google Earth and

[1] http://www.enmap.org/
[2] https://earthdata.nasa.gov/eosdis
[3] https://www.nsmc.org.cn/en/
[4] https://data.globalchange.gov/organization/china-center-resources-satellite-data-application
[5] https://sia.org/news-resources/state-of-the-satellite-industry-report/
[6] https://www.google.com/intl/es/earth/

advanced processing algorithms. At the academic level, we have also seen important efforts in top journals launching multiple special issues devoted to the processing and analysis of remotely sensed big data [1], [6]. Despite these significant advances, the processing of remotely sensed big data still faces significant challenges that we summarize in the following.

1) *Data Integration Challenges:* A unified data standard is needed for heterogeneous remote sensing data integration. This includes uniform data standards, metadata standards, and image standards. However, due to the massive, multimodal, and heterogeneous nature of big remote sensing data, this is a challenging and yet unaccomplished goal.
2) *Data Processing Challenges:* How to design computationally efficient and application-specific data processing and storage techniques (while providing unified interfaces to simplify the access to distributed collections of big remote sensing data) is also a pressing challenge. In a computing system, data transmission is generally the bottleneck due to the limited network bandwidth [8]. Also, the dependence between tasks may introduce ordering constraints, and the optimized scheduling of these tasks may be critical to achieve satisfactory processing performance.

B. Cloud Computing in Remote Sensing

Currently, cloud computing[7] platforms are increasingly being used to process and store remotely sensed big data in distributed architectures [7]. The explosive growth of remote sensing big data has revolutionized the way these data are managed and processed. Still, important challenges remain due to the complex management of multimodal, multispectral, multiresolution, and multitemporal remote sensing data sets, which appears in various formats and/or distributed across data centers. In this regard, cloud computing (based on virtualization technology) offers the potential to integrate computing, storage, network, and other physical resources to build a virtual resource pool where advanced techniques for remote sensing data processing can be developed and deployed. In other words, the cloud provides users with services to integrate data, processing, production, computing platforms, storage, and integrated spatial analysis, so as to provide solutions in different application domains, such as environmental problems, land-use/land-cover, and urban planning. Cloud computing technology also offers advanced capabilities for service-oriented and HPC. The use of cloud computing for the analysis of large repositories of remote sensing images is now considered a natural solution, resulting from the evolution of techniques previously developed for other types of computing platforms, such as commodity clusters or grid environments [12].

Quite surprisingly, there are not too many works yet describing the use of cloud computing infrastructures for the implementation of remote sensing data processing techniques. This is partially due to the lack of open repositories containing labeled remote sensing images for public use, a situation that is now changing due to the initiatives, such as BigEarthNet.[8] Also, NASA and ESA are now providing large distributed repositories of remote sensing data for open use by the scientific community (e.g., the Sentinel program).[9] Due to the availability of such repositories, the development of techniques based on cloud computing for distributed processing of remote sensing images has become a very timely research line.

A relevant question at this point is whether cloud computing can become a *de facto* architecture for remotely sensed data interpretation in future years. Our belief is that, by virtue of its elasticity and high transparency levels, cloud computing offers a truly unique paradigm for big remote sensing data processing, in which computational resources can be accommodated in the form of ubiquitous services on-demand, on a pay-per-use basis. In addition, cloud-enabled remote sensing data processing infrastructures and services can now be delivered for large-scale remote sensing data across geographically distributed data centers, which was impossible even with the most powerful compute clusters. As a result, the incorporation of the cloud to big remote sensing data processing initiatives reveals its capacity to deal with the increased computational and storage challenges introduced by modern remote sensing applications, especially when coupled with the powerful new DL algorithms [13] that have been shown to provide an excellent tool for information extraction from scientific data, in general, and from remotely sensed data sets, in particular [14].

C. Article Organization and Contributions

In this article, we provide a review of the most important initiatives that have been developed so far in the use of cloud computing architectures (compared with other HPC solutions, such as commodity clusters or grid computing platforms) for remote sensing data interpretation, with particular focus on DL techniques and their implementations in the cloud. We believe that this review is quite unique in the sense that it provides a completely new flavor with regards to other published works that focus on DL in remote sensing [15]–[17], big remote sensing data [1], [6], or HPC in remote sensing [4], [18], [19]. None of these review papers considered cloud computing as a *de facto* architecture for big remote sensing data processing, which is now a widespread implementation option (in particular, when computationally demanding DL algorithms are involved in the information extraction process). As a result, we believe that this review is necessary given the recent advances in processing strategies, which inevitably

[7]A formal widely accepted definition of cloud computing can be found in [11].

[8]http://bigearth.net/

[9]https://sentinel.esa.int/

led to using DL algorithms in the cloud for the successful processing and interpretation of big remote sensing data sets. The remainder of this article is structured as follows.

1) Section II provides a general overview of techniques for DL in remote sensing data processing and taxonomy of DL architectures that have been widely used in this context.
2) Section III provides a comprehensive review of available approaches for the efficient implementation of remote sensing data processing techniques based on DL algorithms in HPC architectures, including clusters, grids, and cloud computing systems. This section also includes a discussion and some critical observations resulting from the in-depth analysis of the works published so far in these areas.
3) Section IV focuses on cloud computing as the current *de facto* architecture for big remote sensing data processing using DL algorithms. This section first provides some basic concepts about cloud computing. Then, it details some of the most popular frameworks and programming models that have been used for DL-based processing of remotely sensed data and other scientific applications in the cloud.
4) Section V describes the most popular ML and DL libraries and frameworks in cloud computing environments, with a particular emphasis on those that have been already used in remote sensing applications.
5) Section VI provides a case study with a processing example that illustrates a representative technique for DL-based distributed processing of remotely sensed hyperspectral data in the cloud, providing also some suggestions for practical use and exploitation.
6) Finally, Section VII concludes this article with some remarks and hints at plausible future research avenues.

II. DEEP LEARNING IN REMOTE SENSING

A. Remote Sensing Data Processing

Remote sensing technology now provides high-quality data from the surface of the Earth (in terms of detailed resolution, good signal-to-noise ratio, robustness to perturbations, and accurate error corrections). Advanced analysis and interpretation methods are required to extract the most useful information contained in the data. As a result, the remote sensing discipline involves many Earth science disciplines, such as meteorology, geology, or ecology, as well as a variety of engineering skills to properly interpret the huge amount of remotely sensed data provided by a constellation of air/space EO instruments.

Furthermore, a wide variety of remotely sensed data can be obtained from these instruments, where optical imaging systems capturing reflected sunlight are pretty popular due to the rich information that they contain, with different formats and resolutions (spatial, spectral, and temporal), enabling a very detailed and comprehensive assessment of surface properties [20], [21]. For illustrative purposes, Table 1 lists some popular remote sensing instruments that have been or are currently operational.

Table 1 List of Popular Remote Sensing Instruments. GSD: Ground Sample Distance, PAN: Panchromatic, MSI: Multispectral, and HSI: Hyperspectral

Name	Type	Spatial resolution	Spectral bands	Spectral Range
RADARSAT-2 [22]	SAR	1-100 m	1 (C-band)	5.405 GHz
ALOS PALSAR [23]	SAR	6.25-12.5 m	1 (L-band)	1.27 GHz
AirMOSS [24]	SAR	100 m	1 (P-band)	0.43 GHz
TerraSAR-X [25]	SAR	0.24-40 m	1 (X-band)	19.65 GHz
Sentinel-1 [26]	SAR	5-20 m	1 (C-band)	5.404 GHz
LVIS [27]	Laser	20 m	1	1064 nm
EROS-B [28]	PAN	0.7m	1	500-900nm
GAOFEN-1 [29]	MSI	2.0-8 m	5: PAN Blue Green Red NIR	450-900 nm 450-520 nm 520-590 nm 630-690 nm 770-890 nm
GAOFEN-2 [29]	MSI	0.81-3.24 m	5: PAN Blue Green Red NIR	450-900 nm 450-520 nm 520-590 nm 630-690 nm 770-890 nm
IKONOS [30]	MSI	0.82-3.2 m	5: PAN Blue Green Red NIR	526-929 nm 445-516 nm 506-595 nm 632-698 nm 757-853 nm
WorldView-4[31]	MSI	0.3-1.2 m	5: PAN Blue Green Red NIR	450-800 nm 450-510 nm 510-580 nm 655-690 nm 780-920 nm
Sentinel-2 [32]	MSI	10-60 m	13	443-2190 nm
Sentinel-3 OLCI [33]	MSI	300-1200 m	21	400-1200 nm
Landsat-8 [34]	MSI	15-30 m	11	300-12510 nm
MODIS [35]	MSI	250 - 1000 m	36	405-14385 nm
CHRIS [36]	HSI	18-36 m	62	415-1050 nm
AVIRIS [5]	HSI	20 m	224	360-2450 nm
AVIRIS-NG [37]	HSI	0.3-4.0 m	600	380-2510 nm
ROSIS [38]	HSI	1.3 m	115	430-860 nm
CASI [39]	HSI	2.5 m	114	360-1050 nm
HYDICE [40]	HSI	1-7 m	210	400-2500 nm
HYMAP [41]	HSI	5 m	126	450-2500 nm
PRISM [42]	HSI	2.5 m	248	350-1050 nm
EnMAP [43]	HSI	30 m	228	420-2400 nm
HISUI [44]	HSI	30 m	185	400-2500 nm
DESIS [45]	HSI	30 m	180	400-1000 nm
HYPERION [46]	HSI	30 m	220	400-2500 nm
PRISMA [47]	HSI	30 m	237	400-2500 nm
SHALOM [48]	HSI	10 m	241	400-2500 nm

In fact, optical remote sensing data play an important role in many different activities [49]. On the one hand, PAN, standard RGB, and multispectral products often exhibit impressive spatial resolution, which strongly facilitates the detection of contours, textures, and structures within the scene. On the other hand, although the spatial resolution is partially sacrificed, hyperspectral instruments collect hundreds of narrow and near-continuous bands ranging from the VNIR to the SWIR wavelength regions, providing very detailed spectral signatures of the materials covered by each pixel, thus allowing more accurate and detailed analysis of the spectral features available in the data [50].

The advanced products resulting from the analysis and interpretation of remotely sensed data sets allow for the implementation of long-term development strategies in several fields, such as precision agriculture [51], urban planning [52], or desertification monitoring [53], among others.

The rich and detailed information contained in remote sensing data needs to be adequately extracted and processed to be consequently exploited at the user level. In this regard, there is a strong demand for accurate and

Fig. 1. *Remote sensing classification can be tackled at three different levels: pixel-level (left), object-level (center), and scene-level (right).*

computationally efficient analysis techniques, which can be categorized considering different criteria. In particular, according to their purpose [54], we can classify available techniques into the following groups.

1) *Restoration and denoising methods* manage data corruption and anomalies introduced during the acquisition process that may significantly degrade the quality of the collected data [55], involving radiometric and geometric corrections, or diffuse solar radiation, and management of atmospheric effects.
2) *Enhancement methods* transform the captured data to increase the quality of certain features, making them more suitable to human vision skills to conduct visual analysis. This involves contrast enhancement, super-resolution, and pan-sharpening, for instance.
3) *Transformation methods* modify the scene content in either the spectral or spatial domain for feature extraction, image compression, or filtering purposes, such as the PCA, TCT, vegetation indices, or the WT.
4) *Classification methods* interpret the content of remotely sensed scenes. Three classification levels can be distinguished (see Fig. 1) [56], where pixel-level classification labels each pixel in a scene with a semantic class [57]; object-level classification seeks to recognize the elements present in the scene (by usually combining spectral–spatial features) [58]; and scene-level classification provides a global meaning (i.e., a semantic class) to the entire scene by understanding and interpreting its features [59]. Moreover, subpixel analyses can also be conducted by analyzing the spectral mixtures at each pixel (termed subpixel classification or spectral unmixing) [18], [60], [61].

Focusing on data transformation and classification approaches, ML and DL methods have provided a wide range of processing algorithms for both regression and classification of complex nonlinear systems [14], [16], [62], [63], implementing promising learning paradigms to derive information from the data. These methods range from purely unsupervised strategies to supervised ones, with a vast collection of semisupervised and hybrid-based methods in between [64]–[67].

For instance, k-means clustering is a popular unsupervised method that groups similar samples together by exploring similarity measures and is able to discover underlying patterns [68]. On the contrary, the kNNs usually explores the similarity between samples in a supervised way [69]. Also, DTs [70] and RFs [71] are supervised methods, where RFs develop multiple trees from the randomly sampled subspace of the input sample and then combine the output through a voting/maximum rule. In addition, the MLR [72], GMMs [73], and naive Bayes-based (NB) approaches [74] are probabilistic models that analyze the data distribution to conduct their assumptions. HMMs [75] and SVMs [76] are accurate statistical classifiers. Particularly, the SVM is considered an efficient and stable algorithm for high-dimensional data classification. This method learns the decision hyperplane that can best separate training samples in a kernel-included high-dimensional feature space. Finally, ANNs [77] are versatile empirical-modeling algorithms composed of hierarchical layers of neurons that process input stimuli using synaptic weights and transmit their responses through activation functions. As a result, each layer refines the neural responses to the input data, obtaining increasingly abstract representations by adjusting the model weights, which are automatically learned from the data through the forward–backward propagation mechanism to extract the most relevant information. Moreover, ANNs offer a very flexible architecture in which both the number of layers and neurons (and even the shape and direction of the connections) can be established by the programmer. In this sense, the great flexibility and automatic adjustment of neural models (without any prior knowledge about the statistical features of the data) are major advantages that have positioned ANNs as a very attractive approach, creating a sharp contrast to traditional ML techniques, which usually requires careful engineering to extract complex handcrafted features, requiring specific knowledge to recognize the specific regularities present in the data.

Indeed, the study and implementation of ANNs are so extensive that, within ML, the subfield of DL has emerged as a hot topic for signal data processing [13]. Particularly, DL algorithms have gained significant popularity in remote sensing data analysis over the past few years. For illustrative purposes, Fig. 2 shows the total number of papers per year published on this topic and the number of citations received, revealing an exponential increase in recent years. The figure was generated using the Web of Knowledge engine,[10] and the exact search string used (and the date of the query) is specified in the figure caption—where "AB" indicates that the search was conducted in the abstracts of the papers—for reproducibility purposes.

[10]http://www.webofknowledge.com/

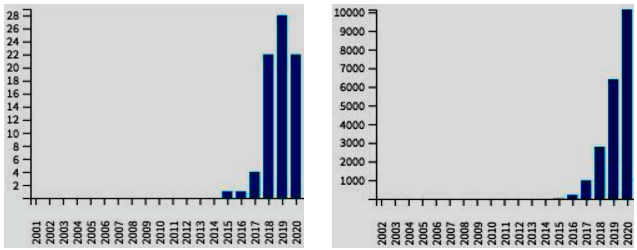


Fig. 2. *Total number of (a) papers per year and (b) citations received by papers in the area "DL in remote sensing." Source: Web of Knowledge. Search string: AB = ("deep learning" AND "remote sensing"). Total number of results: 1621. Date of the query: February 20, 2021.*

A detailed review of available approaches in this area was given in [15], which discusses how DL has been applied for remote sensing data analysis tasks, such as image fusion, registration, scene classification, object detection, land-use and land-cover classification, and segmentation. Different application fields are also covered, including open challenges and directions for future research. Also, the paper [17] focuses on remotely sensed hyperspectral data, providing a comprehensive review of methods for DL-based classification in this field and discussing the strengths and weaknesses of these methods. Accordingly, Section II-B includes a brief taxonomy of the main DL models developed for remote sensing data analysis.

B. Taxonomy of DL Architectures

There is a great variety of deep models due to the great flexibility of existing architectures in terms of topology, data-path connections, and types of layers. In general terms, the scientific community recognizes five models, i.e., SAEs, DBNs, RNNs, and CNNs, as the main architectures, from which a great variety of modified networks have been implemented [17], such as generative adversarial networks (GANs), which have gradually become a mainstream architecture in the field of remote sensing. In the following, we review these DL models.

1) SAEs: The autoencoder (AE) implements an encoder–decoder structure to learn a code representation from the input data in an unsupervised way. It defines an optimization problem that attempts to minimize the reconstruction error $|| \left(\mathbf{W}_d \sigma \left(\mathbf{W}_e \mathbf{X}^T\right)\right)^T - \mathbf{X}||_2^2$ by learning the matrices of bases $\mathbf{W}_d$ and $\mathbf{W}_e$, where $\mathbf{X} \in \mathbb{R}^{N \times B}$ defines the remote sensing data as an input matrix of N samples with B channels (feature space), $\mathbf{W}_e \in \mathbb{R}^{B' \times B}$ comprises the recognition weights of the encoder, which maps the input data to a code/latent representation with $B' \neq B$ features (code/latent space), and $\mathbf{W}_d \in \mathbb{R}^{B \times B'}$ comprises the generative weights of the decoder, which recovers the original feature space by reconstructing the input data. σ acts as an activation function. The SAE deepens the model by stacking several AEs [see Fig. 3(a)], where the AEs of the stacked-encoder (bottom-half) find a series of lower dimensional features, and the stacked-decoder (top-half) performs the opposite function [78].

2) DBNs: The DBN is a multilayer generative model inspired by RBMs. An RBM is a two-layer stochastic network trained to minimize the input reconstruction error in a similar way as AEs (*Gibbs sampling*), where the visible layer deals with the input data and the hidden layer conducts feature extraction, capturing higher order data correlations observed in the visible units while learning a probability distribution over the input data [see Fig. 3(d)]. DBNs take advantage of RBMs, concatenating several pretrained RBMs and refining the full-model parameters through labeled data.

3) RNNs: The RNN [see Fig. 3(b)] retains memory and learns data sequences by introducing loops in its connections. As a result, the neural responses in each step depend on those of the previous step by means of an internal state, which creates a sequential dependence that provides an association between the current and the previous data sample. According to which hidden states are created and how they are managed, three types of RNNs can be distinguished [79]: the simple vanilla RNN, the gate-based LSTM, and the simplified gated recurrent unit (GRU).

4) CNNs: In contrast to other deep models (which were originally implemented with fully connected layers, e.g., the AE), the CNN introduces the convolution layer as a set of locally connected weights that are rearranged in an n-dimensional grid. As a result, the convolution kernels act as feature detection-extraction filters, where neuronal responses are arranged in a feature map that not only indicates the presence of a particular stimulus detected by the kernel (edges, borders, and shapes) but also the location of these stimuli in the spatial domain. This enables abstract and refined spatial relationships to be maintained and extracted within the data through a hierarchical stack of convolution layers [see Fig. 3(c)], which are combined with other layers (activation, normalization, and pooling functions, for instance) to extract elaborate patterns from the raw inputs.

The flexibility of convolution kernels has demonstrated a great potential to extract any kind of feature from the raw data, without applying complex preprocessing mechanisms. Furthermore, the great architectural plasticity in terms of kernel size and grid organization (which produces 1-D, 2-D, and 3-D models), layer connections (direct, residuals, skip, and short-connections), data paths settings, and wide/depth configuration, along with the impressive generalization power and the ability to make strong assumptions about the input data, have established convolution-based models as the most successful and popular deep networks. In fact, these networks represent the state of the art in image processing through

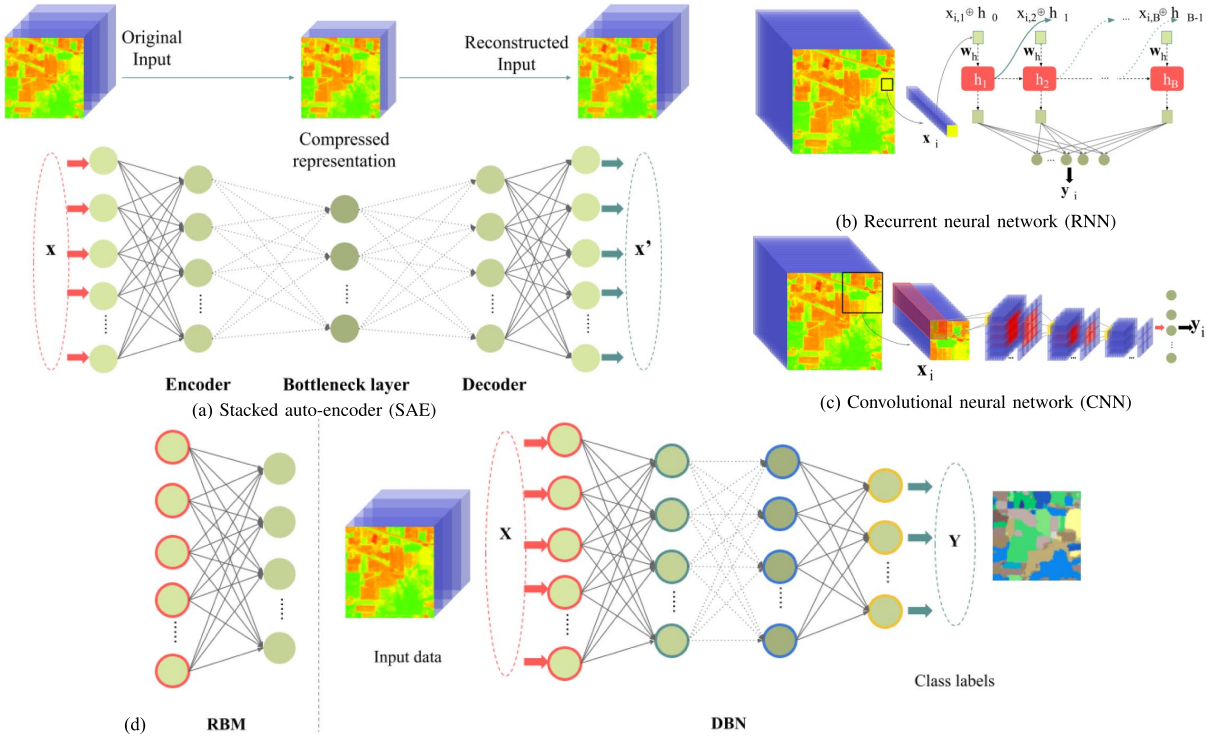


Fig. 3. ***Graphical illustration of traditional deep network architectures applied for processing remote sensing data cubes. (a) SAE optimizes the reconstruction error between its vector input X and its output X′, where the bottleneck layer contains the latent data representation. (b) In the traditional RNN model, the data cube is processed in band-by-band fashion, where the spectral signature of each pixel x_i is processed as a temporal sequence, obtaining, as a result, a hidden state h that works as the model memory. (c) CNN processes 3-D inputs (i.e., the pixel x_i and its surrounding area) by applying multidimensional kernels that act as filters for particular features (borders, shapes, and so on), obtaining a set of feature maps with the neuronal responses to that stimuli. (d) Finally, the DBN is composed of several RBMs that process and reconstruct the input data, mimicking the SAE behavior to learn the probability distribution of the input in an unsupervised fashion (DBN based on RBM, where the visible layer is highlighted with a colored border).***

derived network models, such as residual (ResNets), dense (DenseNets), and capsule-based (CapsNets), among many others [80], [81].

5) GANs: The aforementioned networks work as discriminative models, which maps original inputs to some desired outputs (by learning conditional distributions between them) to minimize a loss function. In contrast, generative approaches (such as GANs) learn the joint probability between inputs and outputs, modeling the data distribution to generate new samples rather than just evaluating the available ones. Thus, GANs (see Fig. 4) model a data distribution from a random noise vector through an adversarial process, where two neural models, i.e., *generative* and *discriminative* networks, are simultaneously trained in the competition (the former to deceive the latter, and the latter to avoid being deceived by the samples generated by the former).

III. IMPLEMENTATIONS

In typical DL models, such as those illustrated in Fig. 3, there are millions of parameters (which defines the model), and large amounts of data are required to learn these parameters. This leads to a computationally intensive process in which the learning step consumes a lot of time. Therefore, it is important to come up with parallel and distributed algorithms that can run much faster and drastically reduce the training times in the context of remote sensing applications. In the following, we provide a description of different parallelization strategies for accelerating DL algorithms in remote sensing applications by resorting to three types of HPC architectures: clusters, grids, and clouds. A description of the main challenges faced by these different architectures, along with a brief comparison among them, is then presented. The section

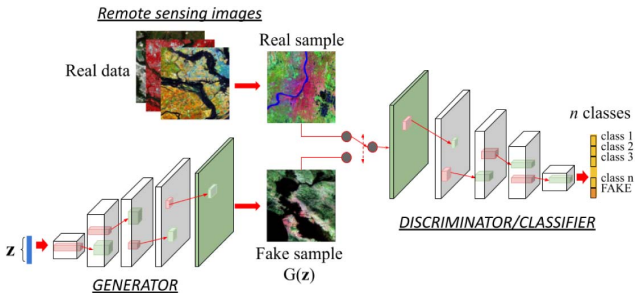


Fig. 4. ***GAN for remote sensing data processing.***

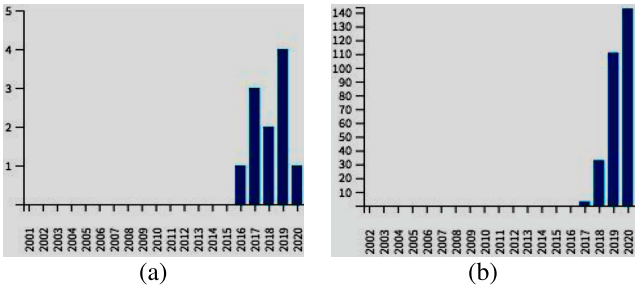


Fig. 5. *Total number of (a) papers per year and (b) citations received by papers in the area "DL in remote sensing using cluster computing." Source: Web of Knowledge. Search string: AB = ("deep learning" AND "remote sensing" AND "cluster" AND "comput*"). Total number of results: 11. Date of the query: February 20, 2021.*

concludes with a discussion on their potential role in solving remote sensing problems via DL algorithms.

A. Cluster Computing

In this section, we describe some of the most relevant approaches in the recent literature to exploit cluster computer architectures (including GPU clusters) for the efficient interpretation of remote sensing data. Fig. 5 shows the total number of papers published in this area, along with the number of citations received (according to Web of Knowledge). In the following, we discuss some of the most relevant contributions in this field.

As one of the most notable recent developments, Lunga *et al.* [82] implement a RESFlow for improving DL algorithms and allowing them to perform computations on large-scale remotely sensed images. The RESFlow works by dividing the data into homogeneous partitions that can fit simple models in homogeneous (i.e., commodity cluster-based) machines. Despite its cluster-oriented nature, RESFlow uses Apache Spark (a tool that has been widely used in cloud computing, as described in Section IV) to accelerate DL inference. The RESFlow incorporates a strategy to optimize resource utilization across multiple executors assigned to a single worker. The framework invokes DL inference at three stages: during deep feature extraction, deep metric mapping, and deep semantic segmentation. Resource sharing in GPUs is adopted to achieve a fully parallelized pipeline for all execution steps.

RESFlow uses Apache Spark, but there are multiple options to distribute training over cluster computing implementations. Other options include TF,[11] PyTorch,[12] or Horovod.[13] These frameworks provide multiple benefits. For example, Pytorch includes multiple extensions (as NVIDIA Apex[14]) to enable streamline mixed precision with distributed training, and Horovod employs efficient inter-GPU communications. Similar to Apache Spark, some cluster computing approaches take advantage of Kubernetes[15] (k8s) architecture workflow, which allows for the deployment automation, scaling, and management of ML/DL applications, as described in [83].

The work in [84] exemplifies the unique advantages provided by parallel computing environments and programming techniques to solve large-scale problems, such as the training of classification algorithms for remote sensing data. Specifically, the authors demonstrate that the training of deep CNNs can be efficiently implemented using cluster computers containing a large number of GPUs. The obtained results confirm that parallel training can dramatically reduce the amount of time needed to perform the full training process, obtaining almost linear scalability in a cluster of GPUs without losing any test accuracy.

At this point, we emphasize that some works do not need to exploit clusters of GPUs to conduct the desired calculations. For instance, in the work [85], the authors resort to a shared memory system with only four GPUs to develop an MTFC of a CNN for remotely sensed hyperspectral image classification. The authors first develop a PNPE that generates 3-D cube samples from the input data automatically. Then, they perform a series of optimizations in the MTFC, such as task division, the fine-grained mapping between tasks and GPU thread blocks, and shared memory usage reduction. To further improve the training speed, the authors exploit CUDA streams and multiple GPUs to train minibatches of data samples simultaneously. The MTFC is shown to outperform popular ML frameworks, such as Caffe[16] and Theano,[17] while offering the same level of classification accuracy.

The paper [86] provides several parallelization approaches for DNNs, taking into account network overheads and optimal resource allocation, since network communication is often slower than inter-machine communication (while some layers are more computationally expensive than others). Specifically, the authors consider a multimodal DNN architecture and identify several strategies to optimize performance when training is accomplished on Apache Spark (this framework will be described in detail in Section IV). The authors compare their newly developed architecture with an equivalent DNN architecture modeled after a data parallelization approach. The experiments in the paper reveal that the way in which the model is parallelized has a very significant impact on resource allocation and that hyperparameter tuning can significantly reduce network overheads.

Other relevant developments in this area include the paper [87], which presents parallel versions of DL techniques for dimensionality reduction of remotely sensed images, also implemented in an Apache Spark cluster. The

[11] https://www.tensorflow.org/
[12] https://pytorch.org/
[13] https://horovod.ai/
[14] https://nvidia.github.io/apex/
[15] https://kubernetes.io/es/
[16] https://caffe.berkeleyvision.org/
[17] https://github.com/Theano/

paper [88] presents an improved version of the aforementioned development that scales even better in clusters of GPUs. The paper [89] presents a parallel and distributed DL-based spectral unmixing algorithm for remotely sensed hyperspectral data, again using Apache Spark for the implementation on a cluster computer.

B. Grid Computing

Although many parallel systems are inherently homogeneous, a most recent trend in HPC systems is to use highly heterogeneous computing resources, where the heterogeneity is generally the result of technological advancement in the progress of time. With increasing heterogeneity, grid computing emerged as a premier technology that could facilitate the processing of remote sensing data in heterogeneous and distributed computing platforms.

Although the grid has recently evolved into architectures with more quality of service, such as the cloud, there were several reasons for using grid computing for remote sensing image processing when the first grid-oriented architectures appeared. First and foremost, the required computing performance may not be available locally. Also, the required performance may not be available in just one location, with a possible solution being cooperative computing. Last but not least, the required computing services may be only available in specialized centers, and in this case, the solution is application-specific computing. This led to the development of some grid-based approaches that are now mostly transitioning into cloud computing implementations, as will be described in Section III-C.

For illustrative purposes, Fig. 6 shows the total number of papers published in this area, along with the number of citations received. In the following, we discuss some of the most relevant contributions focused on using grid computing platforms for solving remote sensing problems via DL algorithms.

The GEOGrid project was one of the first DL-oriented initiatives aimed at providing an e-Science infrastructure to the remote sensing community. It is specifically developed to integrate a wide variety of remote sensing data sets and is accessible online as a set of services.[18]

The platform called G-POD[19] was the first one to provide a grid-based environment for processing satellite images provided by ESA, offering several image processing services and DL algorithms mainly intended for environmental studies. G-POD has been successfully applied in real applications, such as flood area detection.

As an add-on to this tool, the platform for satellite imagery search and retrieval, called GENESI-DR [90], offers an advanced interface for digital data discovery and retrieval, where the original images are processed using G-POD facilities (comprising some DL algorithms). The ultimate goal of GENESI-DR was to build an open-access

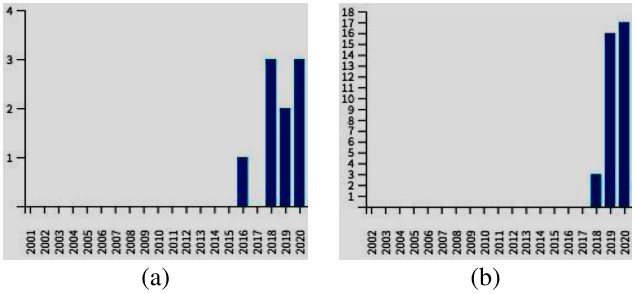


Fig. 6. *Total number of (a) papers per year and (b) citations received by papers in the area "DL in remote sensing using grid computing." Source: Web of Knowledge. Search string: AB = ("deep learning" AND "remote sensing" AND "comput*" AND "grid"). Total number of results: 9. Date of the query: February 20, 2021.*

service to digital repositories focusing on fast search, discovery, and access to remotely sensed imagery in the context of postdisaster damage assessment. Once a disaster alert has been issued, response time is critical to providing relevant damage information to analysts and/or stakeholders. In this regard, GENESI-DR provides rapid area mapping and near real-time orthorectification web processing services to support postdisaster damage needs.

Also, the GiSHEO[20] platform (on-demand Grid services for training and high education in EO) addresses an important need for specialized training services in EO. Solutions were developed for data management, image processing service deployment, workflow-based service composition, and user interaction, with particular attention to services for image processing (able to exploit free image processing tools, along with some DL techniques). A special feature of the platform is the connection with the GENESI-DR catalog, which provides plenty of remote sensing data sets for free.

To conclude this section, it is important to emphasize that the CEOS,[21] an international coordinating body for spaceborne missions focused on the study of the Earth, maintains a working group on information systems and services, with the ultimate goal of promoting the development of interoperable systems for managing EO data internationally. In this regard, several grid platforms have greatly benefited from CEOS standards when developing grid-based tools for accurate interpretation of remotely sensed data [7].

C. Cloud Computing

Cloud computing solutions represent an evolution of grid-based approaches and exhibit the potential to manage and process vast amounts of remotely sensed data in fault-tolerant environments by interconnecting distributed and specialized nodes. This strategy can significantly reduce the processing costs often involved in grid computing,

[18] https://www.geogrid.com/
[19] https://gpod.eo.esa.int/
[20] http://cgis.utcluj.ro/projects/gisheo
[21] https://ceos.org/

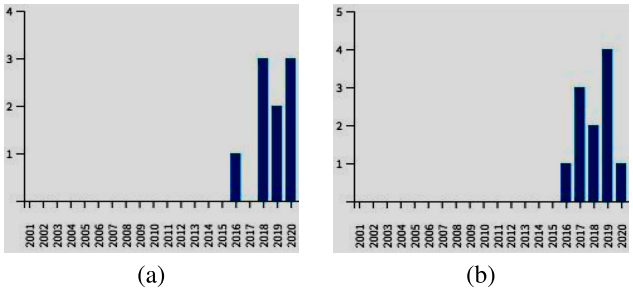


Fig. 7. *Total number of (a) papers per year and (b) citations received by papers in the area "DL in remote sensing using cloud computing." Source: Web of Knowledge. Search string: AB = ("deep learning" AND "remote sensing" AND "comput*" AND "cloud"). Total number of results: 78. Date of the query: February 20, 2021.*

leading to natural and cheap solutions for remotely sensed data processing. For illustrative purposes, Fig. 7 shows the total number of papers published in this area, along with the number of citations. In the following, we review some of the most significant contributions based on cloud computing for solving remote sensing problems via DL techniques.

One of the most relevant works addressing the implementation of DL algorithms for remote sensing data analysis in the cloud was presented in [91], in which the authors introduced a new cloud-based technique for spectral analysis and compression of hyperspectral images. Specifically, the authors provide a cloud implementation of the AE, a popular deep network for non-linear data compression. Apache Spark (described in detail in section IV) serves as the backbone of the cloud computing environment by connecting the available processing nodes using a master–slave architecture. The obtained results indicate that cloud computing architectures offer an adequate solution for compressing and interpreting big remotely sensed data sets.

The paper [92] proposes an acceleration method for hyperspectral image classification that exploits scheduling metaheuristics to automatically and optimally distribute the workload across multiple computing resources on a cloud platform. A representative DL-based classification processing chain is first distributed and implemented in parallel based on the MapReduce mechanism (described in detail in Section IV) on Apache Spark. The optimal execution on Spark is further formulated as a divisible scheduling framework that takes into account both task execution precedences and task divisibility when allocating the divisible and indivisible subtasks onto computing nodes. The scheduling results provide an optimized solution to the automatic processing of big hyperspectral data on cloud environments. The experimental results demonstrate that this approach can achieve significant speedups in the classification of hyperspectral imagery on Spark, obtaining also significant scalability with regards to increasing data volumes.

The paper [93] exploited the idea that state-of-the-art DL-based algorithms and cloud computing infrastructure have become available with a great potential to revolutionize the processing of remotely sensed images. Specifically, their study evaluated, using thousands of images obtained over a 12-month period, the performance of three ML and DL approaches (RFs, LSTMs, and U-Nets). The DL algorithms (LSTMs and U-Nets) were implemented using the TF framework (described in detail in Section IV), while the ML-based RF utilized the Google Earth Engine platform. The study concluded that, although the use of ML/DL algorithms depends highly on the availability of labeled samples and the generalization of these methods still presents some challenges, algorithms based on ANNs can still be used in the cloud to map large geographic regions that consider a wide variety of satellite data formats.

The paper [94] uses cloud computing to make global-oriented spatiotemporal data simulations using the OpenStack management framework (described in detail in Section IV). This is accomplished by resorting to DGGSs, designed to portray real-world phenomena by providing a spatiotemporal unified framework on discrete geospatial data structures, along with a DL-based algorithm to address the challenges resulting from big remote sensing data storage, processing, and analysis.

The paper [95] presents a new parallel CBIR system from remotely sensed hyperspectral image repositories, implemented on a cloud computing platform. The method exploits information from spectral unmixing [96] and DL to accurately retrieve hyperspectral scenes. To this end, the authors implement a distributed and DL-based unmixing method that operates on a cloud computing environment. In addition, they implement a global standard distributed repository of hyperspectral images equipped with a large spectral library in a SaaS mode (this concept will be described in detail in Section IV), providing users with big hyperspectral data storage, management, and retrieval capabilities through a powerful web interface. The parallel unmixing process is then incorporated into the CBIR system to achieve a highly efficient unmixing-based content retrieval system.

Other important contributions in this area include the paper [97], which proposed a model that facilitates the utilization and performance of Apache Spark algorithms in cloud environments. Also, the paper [98] presents the architecture of the ICP data mining package, a distributed tool for the analysis of remotely sensed data. The paper [99] proposes an extension of Apache Hadoop (described in detail in Section IV) that executes operations for processing remotely sensed images (including DL methods) in a highly distributed and efficient way. The paper [100] proposes a highly scalable and efficient segmentation model for remotely sensed images, capable of segmenting very high-resolution imagery with DL algorithms. The paper [101] proposes a method for DL-based cloud computing based on image sampling, which

Table 2 Comparison Between Cluster, Grid, and Cloud Architectures

	Cluster computing	Grid computing	Cloud computing
Scalability	Low	Low	High
Elasticity	No	No	Yes
Heterogeneity	No	Yes	Yes
Compute capability	Cluster dependant	Grid dependant	On-demand
Business services	No	No	Yes
Private service cost	Medium	High	On-demand
Public service available	Yes	Yes	Yes
Resource handling/allocation	Centralized	Distributed	Both
Job queuing	Yes	Yes	No

models the remotely sensed data set to be processed as a streaming service and divides it with a Voronoi diagram. The paper [102] describes a Java software based on the MapReduce model for handling and processing remotely sensed images using DL methods. The paper [103] presents a deep method for storing images using MapReduce. The paper [104] also uses a MapReduce framework for DL-based parallel processing of remotely sensed data through Apache Hadoop. The work [105] describes a set of requirements to achieve a generalized and integrated EO information system and the associated (real-time and off-line) data processing techniques based on DL. The paper [106] presents a new DL-based approach for distributed processing of large-scale satellite images in the cloud. The paper [107] presents a distributed spatiotemporal indexing architecture implemented on the cloud and a distributed DL-based algorithm for improved spatial–temporal queries. The paper [108] discusses the requirements of overlapping data organization and proposes two extensions of the HDFS—described in Section IV—and the MapReduce programming model for dealing with remotely sensed data. The paper [109] describes a DL framework for the efficient analysis of large image volumes, which processes, daily, the data obtained by NASA's EO-1 satellite.

D. Challenges and Comparison

The different distribution perspectives described in the three previous subsections exhibit numerous differences. As a summary, Table 2 provides an overlook of the main similarities and differences between the discussed strategies.

Regarding the cost, there are initiatives that provide free computing platform services for the scientific and research communities. An example is PRACE,[22] aimed at high-impact scientific research and engineering development across different disciplines. Also, there are specific projects for different distributed computing approaches. Condor[23] was created for research and education purposes. EGI-InSPIRE[24] was created by the European Commission for the benefit of the scientific communities within the European Research Area to exploit grid infrastructures. This project is also available for cloud computing.

[22]https://prace-ri.eu/
[23]http://www.cs.wisc.edu/condor/condorg
[24]http://www.egi.eu/projects/egi-inspire/

Usually, cloud computing has been identified as a distributed service, which is not entirely true. Alternatives are UCI[25] or OpenNebula,[26] which brings flexibility, scalability, and simplicity for cloud computing management. In addition to these projects, the XSEDE[27] ecosystem and the EGI[28] (European Grid Infrastructure) provide different cost-free alternatives.

One of the most important features of the cloud (and one of the main reasons for its popularity) is the elasticity and scalability of its architecture. Since cluster/grid architectures are limited to available hardware resources, cloud computing offers the possibility to increase such resources by resorting to the elasticity property, i.e., using resources from different infrastructures. This leads to an increase in the heterogeneity of computing and communication resources, and to the use of both centralized and distributed resource handling and allocation.

Finally, another relevant feature of cloud computing is that the infrastructure does not need to use a job queue for managing executions from different users. This significantly reduces the waiting times for the execution of jobs that are needed in other architectures, such as clusters and grids.

E. Discussion

In this section, we discuss some important aspects identified after the systematic review conducted in the previous subsections, in an attempt to answer relevant questions, such as the role of parallel and distributed computing as an efficient tool for solving remote sensing problems via DL algorithms.

In our systematic review, we have found that distributed computing technologies are highly demanded for DL-based data processing when large volumes of remotely sensed data need to be processed. Commodity cluster computers (possibly including hardware accelerators, such as GPUs) [110], grid environments [111], [112], and cloud computing systems [113] have been the most demanded types of HPC platforms for big remote sensing data processing. Recently, cloud computing has become a standard for distributed processing due to its scalability, low cost, service-oriented, and high-performance properties [7]. Therefore, this technology offers the potential to deal with tasks that must be accomplished over large data repositories. As result, cloud computing can be seen as the most natural solution for the analysis of large volumes of remotely sensed data, as well as an evolution of other HPC techniques (such as cluster and grid computing) that correct their limitations and expands their possibilities.

We have also observed that there are comparatively few efforts in the literature aimed at the exploitation of cluster and grid computing infrastructure for the processing

[25]https://code.google.com/archive/p/unifiedcloud/
[26]https://opennebula.io/
[27]https://www.xsede.org/
[28]https://www.egi.eu/services/cloud-compute/

of remote sensing images compared to cloud computing implementations. In fact, we have noticed that the cloud is now in clear expansion and routinely used to solve remote sensing problems (particularly those involving DL algorithms). This is because of the popularity of the programming languages and frameworks available for implementing DL-based algorithms in the cloud (some of these tools will be described in detail in Section IV).

Another important observation arising from our literature review is that the most widely used tool for remote sensing data processing in cloud computing environments is Apache Hadoop [114], described in mode details in Section IV) [115] although recently Apache Spark [116] has also become a reference tool. The main difference between Spark and Hadoop is the fact that the former distributes the data in RDDs [117] that can be managed more efficiently. As a result, the speedup that Spark can achieve with respect to Hadoop is very high. In addition, Spark provides an ML/DL library called *MLlib* [118]—described in Section IV—that operates in a distributed and parallel manner, so that it provides very good performance with big remote sensing data. Although Hadoop has been widely used in the past, Spark is now a standard due to its speed and better memory usage. Our study also shows that most researchers take advantage of existing cloud computing frameworks when dealing with remote sensing data, rather than developing new ones.

In Section IV, we focus on cloud computing as a *de facto* paradigm for distributed processing of remotely sensed data sets and describe the most widely used frameworks and programming languages that have been adopted in this context (some of them already mentioned in this section), with a particular emphasis on the availability of DL-oriented tools.

IV. FOCUS ON CLOUD COMPUTING

This section first introduces some basic concepts about cloud computing and then glances at the delivery models, available execution frameworks, and programming models supporting this technology (with a particular emphasis on the tools that have been specifically used in remote sensing applications).

A. Cloud Computing Basics

Advances in distributed computing technologies, together with the high amount of data generated and consumed by a growing number of devices (including remote sensing instruments), have leveraged the adoption of emergent cloud computing technology. Nowadays, cloud computing has become a suitable model to cover a wide range of users' needs, including data analytics, data mining, remote sensing, social media, and other computational and data-intensive applications. Cloud computing is a model that provides users with ubiquitous, on-demand access to remote hardware and software resources in the form of services. This model has become a cost-effective solution that usually reduces initial investment, management, and maintenance costs with respect to previous clusters and grid technologies. Cloud computing provides elastic computing, storage, and networking payment services. The term elastic refers to the ability of the model to dynamically adapt to user scalability and variable workload necessities [119], [120].

At the core of cloud computing is the *virtualization technology* [121], [122]. Virtualization abstracts the underlying physical resources, such as computing, storage, and networking, as a set of virtual resources, typically enclosed in the form of isolated instances called *VMs*. Such modular design enables some advantageous features, as replication of data instances, fault tolerance, security (by limiting the interaction between modules), and migration of instances [123].

Cloud computing determines how the virtualized resources are allocated, deployed, and delivered to users. In this sense, delivery models are structured in three basic categories that describe the form in which users access the resources [124]:

1) At the higher level of abstraction is *SaaS*, in which the user accesses to end applications usually developed, managed, and maintained by the cloud provider in its cloud infrastructures. Users access these applications via web interfaces. A common example is a CBIR system for remote sensing data repositories.
2) *PaaS* refers to the provision of development full stacks, including OSs, libraries, management, and monitoring tools. The users accessing those services are usually software developers with limited control over the underlying infrastructure, which is managed by cloud providers.
3) The *IaaS* model lies at the lower level of abstraction and allows users to manage an elastic infrastructure composed of a set of computational, storage, and networking virtual resources. The users develop and deploy their applications on such virtual resources and manage the infrastructure.

Without neglecting its benefits, cloud computing faces an important set of challenges. The most significant is *security*, in the sense of both ensuring the access to the data exclusively by authorized users and systems (privacy), and maintaining the integrity and availability of the data distribution (even geographically) and replication across different locations (e.g., as in different remote sensing data centers). *Performance* is an additional key factor impacting the adoption of cloud computing, especially by scientific applications that usually execute on tightly coupled high-performance clusters.

Clusters and clouds have different design goals and features [125]. While the main goal of clusters is performance (supported by dedicated parallel computing resources connected with minimal latency and stable high bandwidth networks), the goal of clouds is to make available on-demand virtual resources in an elastic platform at a

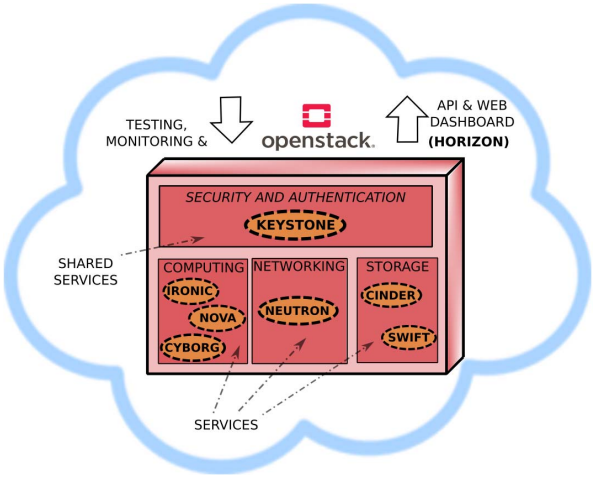


Fig. 8. ***OpenStack main computing, networking, and storage components.***

reasonable cost. Hence, technical management issues—derived from the shared use of physical resources by VMs and multiple users, dynamic on-demand scalability, data movement, virtualization overhead, and workload balance in the presence of heterogeneity—impact the performance of applications deployed on the cloud infrastructure.

B. Deployment Frameworks

Henceforth, we adopt the perspective of users of an IaaS model, allocating a pool of virtual resources connected by networking services to deploy and run scientific applications. As an effective management framework to deploy and run the applications, we highlight OpenStack[29] although there are other open-source frameworks, such as Apache Cloudstack[30] and OpenNebula that offer similar features. In the following, we review the functionality of several of the multiple services available in OpenStack.

Openstack is oriented to manage the complete life cycle of an IaaS cloud system composed of a large number of virtual resources for computing, storage, and networking. The Openstack design architecture is highly modular. Each independent module implements a specific service and exposes a well-defined API, making the system extensible and able to support the integration of third-party services.

Some of the main software components of the OpenStack framework are outlined next and shown in Fig. 8. We structure them into three categories.

1) *Computing service components* for deploying and managing VM and Linux containers. These facilities are supported fundamentally by the *Nova* compute engine. Furthermore, OpenStack offers a general-purpose management framework component for hardware accelerators (such as GPUs) called *Cyborg*.
2) *Networking service components* with support for different network technologies and equipment. *Neutron* is the main component, and it allows managing SDNs and attaching virtual devices to ports on these networks.
3) *Storage service components*, including *Cinder* component for block storage and *Swift* component that delivers services for securely storing unstructured data as a pool of objects and files.

In addition to the aforementioned computing, networking, and storage services, it is worth noting that OpenStack includes multiple software services for monitoring, development, recovery, databases, orchestration of virtualized resources, workload balance, and more. Among them, for instance, *Keystone* provides services for security and authentication of users and applications. Finally, *Horizon* presents a web interface in the form of a dashboard to manage the virtual resources composing the cloud infrastructure.

C. Programming Models

The term, remote sensing big data, was coined to refer to the increasing need of storing, managing, and processing vast amounts of remotely sensed data, produced at a high rate and in a wide range of formats. Such overwhelm flow of data requires flexible parallel platforms with a large computational capacity and specific programming facilities.

In contrast to cluster computer-centric paradigms, as message passing, big data applications require a data-centric approach, in which a computational task should be deployed in a computational resource as close as possible to the data location. As a result, the movement of data through the network can be minimized. The limited capability of the network bandwidth is a factor that, together with the high volume and heterogeneity of remotely sensed data, highly affects the performance. In the following, we review some relevant programming models and application ecosystems that have been widely used in big remote sensing applications.

1) MapReduce: It is a programming model aimed at developing scalable and robust applications working with large data sets [126], [127]. It was originally developed by Google and, together with the distributed GFS [128], has been successfully used in solving numerous big remote sensing data problems (as described in Section III). This model is particularly suitable for data-centric environments, such as big remote sensing data processing on cloud computing platforms [129]. MapReduce provides the developer with a simple interface based on the *map* and *reduce* functions. An application takes as an input a structured set of key-value data. The *map(k1,v1)* function transforms the input to an intermediate set of key-value pairs in the target domain (k2,v2). The MapReduce library combines the intermediate values (v2) on a per-key basis. Finally, the *reduce(k2, list(v2))* function operates on the

[29]https://www.openstack.org
[30]https://cloudstack.apache.org

list of values corresponding to each target key to generate the output. Usually, both *map* and *reduce* are executed by parallel tasks deployed across a set of computational resources. The model interface abstracts the complexities of the execution of the remote sensing application in the specific platform and leaves the underlying details to the implementation.

MapReduce implementations are ultimately based on the master–worker approach, in which a *master* process manages the automatic parallelization and scheduling of the tasks on the computational nodes and coordinates their execution. It partitions the remotely sensed data to be processed by each *map* and *reduce* tasks and schedules those tasks on computational resources as close as possible to the data to be processed. To achieve this, it relies on GFS that provides the locations of the blocks of data to be processed. This design improves data locality and minimizes the data transfers through the network, hence improving the performance.

Besides, MapReduce combines the output intermediate files of the *map* tasks and combines the data according to the output keys. This intermediate stage tackles the complexity of a high amount of communication and coordination tasks, which are hidden to the developer. Finally, the implementation delivers processed data chunks to the *reduce* tasks. Furthermore, MapReduce implementation promotes fault tolerance mechanisms to detect and reexecute tasks when necessary. In this sense, the communication between the deployed tasks is achieved using intermediate files.

2) Hadoop Ecosystem: Hadoop[31] has been widely used to parallelize remote sensing data processing tasks (see Section III). It is a popular open-source and scalable big data software framework based on the MapReduce paradigm, the HDFS, and YARN [115] resource manager.

HDFS basic functionality is similar to that of GFS. Data are split up into blocks of fixed size and replicated across several nodes to ensure fault tolerance and availability. Its implementation follows a master–worker approach. A *Namenode* process manages metadata (such as block mapping information) and delivers that information to the MapReduce library when requested. *DataNode* processes execute in each virtual resource and effectively store data blocks and provide data reading and writing services to applications.

MapReduce functionality is implemented by a *JobTracker* process, which receives job requests and schedules job tasks to different nodes. Each node is controlled by a *TaskTracker* process, which monitors the execution and reports to the *JobTracker* if a problem appears. In such a case, *JobTracker* resubmits the involved tasks to the same or a different *TaskTracker*. In this sense, Hadoop decouples the MapReduce programming model and the associated resource management. YARN delivers the former services. Its internal architecture is based on three main components.

1) The first component is the global per-cluster *ResourceManager* process, which accepts job submissions and allocates resources for the application. It is responsible for scheduling the application in the available resources.
2) The second component is the *NodeManager*, a per-node process. It is responsible for the execution of the tasks assigned to its node.
3) Finally, the *ApplicationMaster* is a per-application process that monitors the application necessities and their status along their lifecycle, negotiating resources with the *ResourceManager*.

The Hadoop framework has been enhanced with multiple tools and services forming the so-called *Hadoop Ecosystem*, which has been exploited in a variety of remote sensing applications (see Section III). It includes relational databases managers (as *Hive*), NoSQL databases (as *HBase*, a column-oriented distributed database running on top of HDFS), distributed ML/DL and linear algebra solvers (as *Mahout*), real-time facilities (*Storm*), efficient alternatives to the MapReduce programming model, and utilities for the orchestration of the services and components (*ZooKeeper*).

3) Apache Spark: Originally developed at UC Berkeley, Apache Spark [116], [130] was designed to gain velocity in the processing of big data. Although the MapReduce model adequately adapts to a large number of applications as highly parallel batch jobs, it incurs significant latency in both interactive applications and in those with an iterative pattern of execution, in which the same data sets need to be continuously reloaded from the file system.

In this respect, one of the most relevant features of Spark is its ability to support persistent data in memory, which greatly benefits performance. This feature is implemented in the run-time system of Spark, known as *Spark Core Engine*. In addition to this module, the Spark ecosystem includes a set of utilities, as *Spark SQL*, which allows managing structured and semistructured data organized in columns, known as DataFrames, the *Spark Streaming* module for performing real-time processing on data streams (ideal for remote sensing applications with real-time constraints, as described in Section III), and the *MLlib* library that includes distributed ML and DL algorithms.

Spark follows a master–worker execution model, with a *driver* node acting as the master and a set of worker nodes. The execution model is shown in Fig. 9. An application submitted to Spark starts its execution in the context of the driver node. The application creates a *SparkContext* object that transforms the sequence of operations described in the main program into an execution plan. The execution plan is represented as a DAG, in which nodes are data elements and edges are operations on such data. SparkContext splits up the DAG in stages to be executed by tasks. Then, SparkContext negotiates the acquisition of resources with

[31] https://hadoop.apache.org

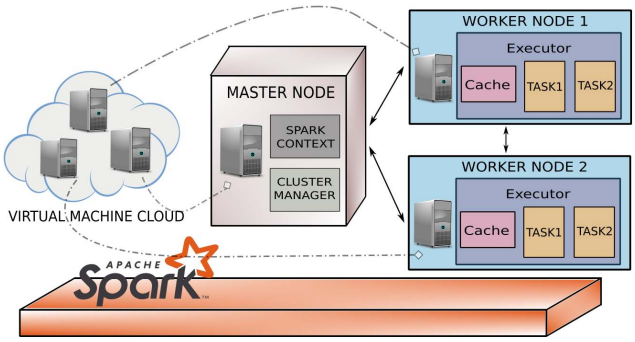


Fig. 9. ***Apache Spark execution model in a cloud computing platform.***

the *Cluster Manager*. While the normal form of executing a Spark application is to use its own native cluster manager, Spark is able to run on Hadoop clusters on top of the YARN resource manager. In any case, after the resources have been obtained, SparkContext launches an *Executor* process in each worker node. Based on the execution plan, SparkContext schedules tasks to worker nodes and coordinates its execution. Conversely, Executors are in charge of effectively executing tasks assigned to its worker nodes and providing access to data.

The *RDD* [117] is the main abstraction supporting the Spark model. An RDD is a read-only collection of objects partitioned and distributed across the worker nodes. The assignment of tasks to worker nodes considers data-locality, that is, the availability (or closeness) of the data to be processed by the task in the RDD partition assigned to the work node. An important feature of RDDs is that the data can be cached in memory and, hence, reused in recurrent parallel operations with minimum overheads. Furthermore, the RDD is a fault-tolerant data structure. In this sense, any operation on an RDD object is logged in such a way that, in the case of a node failure, the RDD can be reconstructed using the operation *lineage*. Because lineage dependencies become large and their management is time-consuming, users may decide to establish checkpoints in the execution. In addition, the immutable nature of the RDD objects benefits another fault-tolerance mechanism, i.e., the execution of backup copies (duplicates) of running tasks if failed or straggler tasks are detected.

RDDs can be created from data structures in memory or in the file system and also using data obtained from any Hadoop service, including the HDFS or databases as HBase. In addition to the mechanisms for creating RDDs, Spark includes a set of coarse-grained operations to process RDD data sets. These operations are structured in two types.

1) The first type is given by *transformation* operations that apply a function to an RDD and generate a transformed RDD data set as a result. In turn, transformations are classified into *narrow*, which involves data located in the worker node where the task executes the operation, and *wide*, which involves data across multiple worker nodes, and therefore, the required data are copied from other partitions. The movement of data is coordinated by the driver. Examples of transformation operations defined in the API are *map*, *filter*, *groupbykey*, and *reducebykey*.
2) The second type of operations is called *Actions*. Actions are operations that retrieve non-RDD values (as statistical or processed values) from RDDs, and their value is returned to the driver program. Examples of actions are *count*, *collect*, *reduce*, and *foreach*. Some of them take as an input parameter a function to be applied to the data.

Transformations are *lazily* executed, in the sense that a sequence of transformations is effectively executed when an action is performed on the transformed RDD. This mechanism improves performance in cases in which only final results (and not intermediate results of the sequence of transformations) are transferred to the driver program. Nevertheless, by default, each transformed RDD is recomputed each time an action is executed on it. The persistence mechanism of Spark allows keeping in memory the data, improving the performance of recurrent operations on the RDD (this is particularly beneficial for image processing operations involving sliding windows or kernels, which are very popular in many remote sensing applications).

Moreover, wide transformations are inefficient operations in the Spark model, consisting of independent tasks operating on their own assigned RDD partitions, because such operations require data movement. Every time a task executes an operation on remote data, SparkContext coordinates the disk I/O and network transmissions between the involved nodes. This costly procedure is called *shuffle*. The persistence capability highly improves the performance of the shuffle operations by caching in memory the RDD data that are going to be reused in wide transformations, which is also popular in multiscale image processing operations adopted in many remote sensing applications (see Section III).

Finally, Spark includes two additional mechanisms to avoid recurrent copies of shared data between the worker nodes, called *shared variables*. The first one is called *broadcast*, and it allows to diffuse a set of values to worker nodes, which will hold a read-only copy of the data. The second one allows to maintain simple (associative) *accumulators* shared across worker nodes.

D. Cloud Computing for Scientific Applications

Scientific applications (including remote sensing ones) exploit the low latency and dedicated resources of clusters to obtain high performance. On the contrary, cloud computing offers elastic multitenancy (resource time sharing) and nonpredictable and unstable network facilities. Nevertheless, there is a great interest in evaluating if cost-efficient and flexible cloud platforms can execute scientific HPC applications at a reasonable level of efficiency.

The paper [131] proposes AzureMapReduce, a decentralized MapReduce implementation for Microsoft Azure cloud infrastructures, and evaluates its (weak-scale) scalability and performance with a remote sensing application. The authors claim that MapReduce applications in cloud infrastructures exhibit comparable performance to MapReduce applications executed on traditional clusters. On the contrary, the work [132] analyzes the performance of the HPC Challenge (HPCC) benchmark [133] on the Amazon EC2 cloud platform and concludes that the performance of general scientific applications on cloud infrastructures is at least one order of magnitude lower than that on clusters and supercomputers. Moreover, the work [134] analyzes the performance of loosely coupled many-task scientific computing applications on four commercial cloud computing provider platforms with the same aforementioned conclusion.

The thorough study in [135] presents a run-time performance comparison of the characteristics of the Amazon EC2 cluster computing instances and a supercomputer. The paper evaluates latency and bandwidth microbenchmarks, HPCC matrix multiplication kernels, NAS Parallel Benchmarks (NPB [136]), and four full-scale remote sensing applications used at NASA. The results show that, in one node, performances are equivalent, while, in several nodes, the network overheads of the cloud computing infrastructure have a huge impact on performance.

The paper [125] evaluates Amazon IaaS services at different levels. The authors execute microbenchmarks to extract raw performance of latency, bandwidth, memory, and processing services. Furthermore, they execute the parallel HPL [137] benchmark to compare cluster and cloud environments. The goal was to identify the advantages and limitations of cloud platforms. They conclude that I/O and network performance differences are the main factors impacting the applications' performance. One of the main drawbacks detected in the cloud is the network infrastructure based on Ethernet, which is often not suitable for the necessities of HPC applications (including remote sensing ones). The paper [123] proposes the HPC2 model that bridges the gap between cluster and cloud platforms with a set of proposals, including using Infiniband as network technology (as it is commonly the case in current remote sensing applications implemented in cloud environments).

There is a consensus in which the performance differences between platforms come from the inherent overheads in virtualization, memory, storage and I/O, and latency of the network infrastructure [138], [139]. Furthermore, the work [140] offers an extensive study of the performance of several cloud providers of public IaaS services: Amazon EC2, Microsoft Azure, GCE, and IBM SL, and it concludes that, indeed, there are substantial differences between the performance of infrastructures of different cloud providers.

To overcome the differences between cluster and cloud platforms, several works maintain that it is not enough to straightforwardly run cluster applications on cloud platforms. This is in contrast with many cloud implementations of remote sensing algorithms described in Section III, which simply run available cluster-based codes in cloud environments. For instance, the paper [141] proposes to slightly transform HPC applications as representative HPC kernel solvers by optimizing computational granularity, which has a high impact on scheduling and communication/computation overlapping. In addition, they propose to transform cloud facilities to use thin VMs and CPU affinity mechanisms. They conclude that, by transforming HPC applications (such as remote sensing ones) to be run in a cloud and making clouds *HPC-aware*, the impact of the latency and multitenancy is significantly reduced. In this sense, the paper [142] proposes to use the MRAP model that extends MapReduce with usual HPC application data access pattern semantics (noncontiguous and fine-grained) while taking advantage of the inherent scalability and fault-tolerance features of MapReduce. This is a promising solution to increase the performance of the MapReduce-based remote sensing implementations described in Section III.

V. MACHINE AND DEEP LEARNING LIBRARIES AND FRAMEWORKS IN CLOUD COMPUTING ENVIRONMENTS

Numerous efforts have been devoted to the development and efficient execution of ML and DL applications on cloud computing infrastructures, not only as optimized libraries and services but also as applications on top of the Spark and MapReduce programming models. In this respect, cloud providers offer several facilities for this challenging task. Among them, Amazon AWS promotes an ML platform called SageMaker[32] to build and train different models, with support for TF and Spark. IBM provides tools for different frameworks, including TF and Keras.[33] GCE also supports the use of TF and provides an infrastructure based on GPU computational devices. Microsoft Azure, on the other hand, bases its services on Kubernetes and allows using accelerators for its ML/DL resources.

Nevertheless, the iterative execution pattern of ML/DL learning applications (in which the same data are recurrently operated, as in many remote sensing data processing algorithms) does not naturally adapt to established cloud computing programming models. Virtualization and network overheads are key factors impacting the efficiency of ML/DL learning applications, which are usually supported on highly optimized computing linear algebra libraries, such as BLASs [143] and high-performance networks and communications based on MPI [144] to achieve high performance.

In addition, ML and DL models have dramatically grown in terms of structural complexity and depth. Training models on huge data sets (such as those involved in remote

[32] https://aws.amazon.com/sagemaker/
[33] https://keras.io/

Table 3 Summary of Parallelization Schemes for Distributed Computing Approaches

Scheme	Cloud computing	Cluster/Grid computing
Data-parallelism	• Master node holds the data. • Data/computation are split. • Model is replicated. • Workers sends results to the master.	• Data are partitioned over executors. • Model is replicated over executors. • Executors share intermediate results.
Model-parallelism	• Master node holds the data. • Data are not partitioned. • Model/computation is split. • Workers sends results to the master.	• Data are replicated over executors. • Model/computation is partitioned. • Executors share layer outputs and results.
Hybrid-parallelism	• Master node holds the data. • Data are partitioned. • Model is split over workers. • Computation depends on data and model workload. • Workers sends results to the master.	• Data are split over executors. • Model is partitioned over executors. • Computation depends on data and model workload. • Executors share intermediate results and outputs.

sensing applications) has become a computationally very intensive (as well as a memory-consuming) task that usually requires several days even using specialized hardware, such as GPUs. To overcome this limitation, several methods for parallel training of ML and DL models have been developed. The parallelization schemes can be structured in three main groups [145], [146].

1) The first type is the *data-parallelism* scheme, in which several replicas of a model are simultaneously trained in different computational devices on disjoints partitions of the remote sensing data set.
2) The second type is called *model-parallelism*, and it is used when a model overcomes the memory capacity of one computational device; then, it has to be partitioned and deployed on several devices.
3) Finally, the last type is the *hybrid-parallelism* scheme that merges data and model-based approaches.

These parallelization schemes can be implemented as multiple distributed computing approaches. Table 3 summarizes the different characteristics of each scheme for cloud and cluster/grid computing approaches.

The rest of the section outlines the main frameworks and libraries offered by cloud providers to face the challenge of efficient training of ML and DL models when processing remotely sensed data on cloud computing infrastructures.

A. Libraries

This section provides an overview of some well-known ML and DL libraries that are used in cloud computing environments to build models efficiently. These libraries have been used in the past to process and accelerate remote sensing applications.

1) Weka: The *Weka*[34] library was developed at the University of Waikato [147]. It is a Java-based open-source library that allows building ML and DL models for several types of algorithms, including classification, clustering, and data mining. A relevant feature of the library is that it is multiplatform and even runs on lightweight devices on top of the Android OS [148]. It supports multiple programming languages with different packages and plugins, such as the DeepLearning4J,[35] the RPlugin,[36] or several Python[37] DL libraries. *Weka* was initially developed to offer a simple and easy-to-use interface. Currently, it provides a distributed version [149] implemented on top of Spark and RDDs.

2) MLlib: The *MLlib*[38] is a distributed ML/DL library that provides model training based on the data parallelism scheme [118]. It was developed in the Scala[39] programming language and supports Java, Scala, and Python programming languages. Its main features are scalability and fast implementation of numerous ML/DL algorithms, as well as linear algebra, statistics, and optimization primitives. It is built on Spark and implements efficient communication primitives for data transmissions performed by a large number of processes and training large models using the data-parallelism scheme. Some communication primitives of special interest are the *broadcast*, which efficiently distributes data over processes training the model, and the tree-structured *aggregation* primitive, which collects processed data avoiding possible bottlenecks. *MLlib* was initially designed to efficiently operate on fully distributed environments, which entails a significant performance advantage with respect to Weka [150].

The execution model of *MLlib* is based on the master–worker paradigm, where the master process acts as a *PS* and maintains a centralized copy of the global parameters of the model. It combines values received from the worker tasks after each training iteration. Tasks deployed on the computational resources of the platform process their assigned data set partitions and communicate results to the PS. The data partitions assigned to each task are processed in *batches*, being the size of each batch a key optimization parameter, which directly affects the resulting accuracy of the model and the efficiency of the training [151], [152].

B. Frameworks

This section discusses different frameworks to develop ML/DL applications in cloud environments.

1) TensorFlowOnSpark: This framework combines salient features of the TF DL library with Spark and Hadoop to provide a scalable ML/DL development and training platform [153]. It supports all TF functionalities, including asynchronous and synchronous training, data, and model-based parallelism schemes, and monitoring with TensorBoard.[40] It also enables distributed TF-based training on cloud computing clusters, with the additional goal of minimizing the amount of code refactoring required to run existing TF applications. In summary, *TensorFlowOnSpark* deploys a Spark cluster on a cloud

[34] https://weka.sourceforge.io/

[35] https://deeplearning4j.org/
[36] https://weka.sourceforge.io/packageMetaData/RPlugin/
[37] https://www.python.org/
[38] https://spark.apache.org/mllib/
[39] https://www.scala-lang.org/
[40] https://www.tensorflow.org/tensorboard

infrastructure and provides facilities for injecting both RDD and HDFS data in the TF models executed by the tasks scheduled in the worker nodes.

2) SparkTorch: This framework is intended to execute code based on the PyTorch library [154] across nodes in a Spark cluster. The distributed training works under a data-parallel paradigm and uses both tree reductions and PS mechanisms to combine partial results from tasks deployed on the cloud platform. There are two main training modes available in *SparkTorch*.

1) The first one is the *asynchronous* training mode, which ensures that the replicated models deployed in nodes are synchronized through each training iteration.
2) The second training mode, called the *Hogwild* approach [155], allows lock-free task accessing to shared memory in order to update parameter values. This mode eliminates the overheads associated with locking. However, in this mode, a task could overwrite the progress of other tasks a risk that the developers claim that could be assumed when the data to be accessed is sparse.

3) BigDL: This framework is also implemented on top of Apache Spark to run DL applications as standard Spark programs [156]. It offers support for large-scale distributed applications and provides efficient processing for data analysis, data injection to neural network models, and distributed training or inference, using a unified pipeline. Before training, the model and RDDs are partitioned and cached in memory across the cloud resources. *BigDL* supports two-parameter synchronization mechanisms. The first one maintains a centralized PS, and the second one uses collective operations as *AllReduce* to combine the parameters computed by tasks. Despite the fact that collective message passing primitives are not particularly suitable for the execution model of a Spark cluster, *BigDL* implements an efficient *AllReduce* algorithm using Spark primitives, allowing for the integration of DL algorithms in cloud computing environments.

It is important to note that the three aforementioned frameworks can use two different communication approaches.

1) The *PS approach*, as illustrated in Fig. 10(a) [157], consists of a centralized architecture where the computational nodes are partitioned into masters and workers. The workers maintain a workload and data partition, while the master maintains the global shared parameters. The workers communicate with the master to share the weights generated at each iteration of the model. The master is responsible for the aggregation of the global weights. In cloud computing environments, additional workers may be added or removed from the execution. This must be handled by the system, so as to switch on any new workers and send to them the appropriate computations and data partitions.

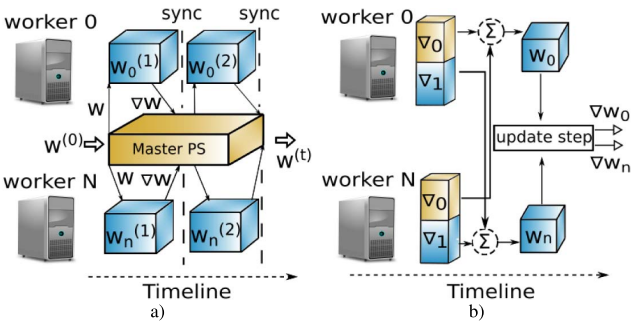


Fig. 10. ***Cloud computing training pipeline in the three discussed DL frameworks (TensorFlowOnSpark, SparkTorch, and BigDL). (a) Parameter server approach. (b) AllReduce-Ring BigDL approach.***

2) The *AllReduce-Ring BigDL approach*, as shown in Fig. 10(b), consists of a decentralized architecture where each Spark task computes its local gradients, dividing the local gradients into N partitions. Each task manages its corresponding parameter partition, which is shuffled to the corresponding task to aggregate gradients and then update the corresponding weights. Then, each task launches a Spark broadcast operation with the updated weights so that these are read before the next step.

A discussion between the aforementioned frameworks and their pros and cons in cloud environments is needed at this point.

1) In terms of applications, all of them provide full integration with Spark ML/DL algorithms. While *TensorFlowOnSpark* provides a large amount of algorithms and applications, *BigDL* includes extensive DL functionalities.
2) The ease of use differs among different frameworks. While the *TensorFlowOnSpark* interface is clear and easy to use, the documentation for *BigDL* is intuitive and provides a comprehensive support for ML/DL algorithms.
3) Attending to the distributed training, *SparkTorch* provides asynchronous and synchronous schemes since *TensorFlowOnSpark* asynchronous PS is highly efficient. We also note that *SparkTorch* is in a premature developing phase compared to the *TensorFlowOnSpark* and *BigDL* implementations. Thus, the latter frameworks still offer notable advantages. An important aspect of *TensorFlowOnSpark* is the creation of checkpoints to recover from failures. These checkpoints are stored in the HDFS by TF. A similar point between these two last frameworks can be found in the monitoring.
4) In terms of scaling and performance, *BigDL* takes a step forward from the other frameworks. This is due to multiple factors. First, it provides extensive documentation to deploy ML/DL algorithms in different providers as EC2. Also, attending to the execution, it provides a synchronous SGD and an optimized

AllReduce in the communication step. Another interesting point is that it can be used only for prediction, and hence, it can load models from different ML/DL frameworks. Finally, the aforementioned frameworks execute powerful long-running tasks, while *BigDL* uses short-running, nonblocking tasks for the model computation.

VI. CASE STUDY

This section presents a case study in which a DNN (implemented on the cloud) is used to process a large hyperspectral remote sensing image. As noted before, hyperspectral data cubes comprise a significant amount of information, which allows us to model the physical characteristics of the observed materials by analyzing the detailed spectral signatures (collected on a pixel-by-pixel basis), which provides rich information for land-cover analysis. When applied to hyperspectral images, classification methods suffer from important processing time and computing/storage constraints, resulting from the extremely high dimensionality of the data. Therefore, the implementation of such classifiers in cloud computing architectures is an effective solution. Here, we use a deep MLP [17] as the distributed classifier and perform classification experiments on a benchmark classification data set widely used in the hyperspectral imaging community.

The remainder of this section is organized as follows. First, we describe our cloud implementation of the MLP classifier. Then, we describe the hyperspectral image used for validation purposes. Then, we provide the characteristics and configuration of the cloud computing platform used for experiments. The section ends with a detailed discussion of our conducted experiments and with some remarks on the practical utility of distributed DL algorithms in remote sensing applications.

A. Distributed Multilayer Perceptron Classifier

Let us denote a hyperspectral data cube as $\mathbf{X} \in \mathbb{R}^{h\times w\times n_{\text{bands}}}$, where each data sample $\mathbf{x}_i$ is of size $h \times w$ (h being the height and w the width in pixels of the image), and each pixel can be denoted by $\mathbf{x}_i \in \mathbb{R}^{n_{\text{bands}}} = [x_{i,1}, x_{i,2}, \ldots, x_{i,n_{\text{bands}}}]$. In an MLP classifier, each layer l performs a data transformation of the weights and the input data ($\mathbf{W}^l$ and $\mathbf{x}_i$) as follows:

$$\mathbf{x}_i^{l+1} = \mathcal{H}(\mathbf{x}_i^l \cdot \mathbf{W}^l + b^l) \tag{1}$$

where $\mathbf{x}_i^{l+1}$ is a feature representation of the input data obtained by the neurons of that layer (l). Neurons are obtained as the dot product between the output from the previous layer neurons plus the bias b (shift parameter) through an activation function, such as the ReLU or the sigmoid function, among others [17]. Hence, the kth feature of the $\mathbf{x}_i^{(l+1)}$ sample can be obtained as

$$x_{i,k}^{l+1} = \mathcal{H}\left(\sum_{j=1}^{n^{l-1}} (x_{i,j}^l \cdot w_{k,j}^l) + b^l\right). \tag{2}$$

Attending to the MLP optimization step, the optimizer tries to obtain the set of parameters $\mathbf{W}$ and $bias$ that minimize the loss error. Hence, a backpropagation step calculates the gradient of the error in order to minimize the final error. At each step, the updating process is defined as follows:

$$W_{t+1} = W_t + \nabla W \tag{3}$$

where $\Delta W = \mu_t \cdot \mathbf{p}_t$, with μ being the learning rate and $\mathbf{p}$ the descent direction of the gradient at time step t. To optimize this operation, traditional methods use the information obtained from the Hessian matrix

$$\mathbf{H}_t \cdot p_t = -\nabla E(\mathbf{X}, \mathcal{W}_t) \tag{4a}$$

$$\mathbf{p}_t = -\mathbf{H}_t^{-1} \cdot \nabla E(\mathbf{X}, \mathcal{W}_t) \tag{4b}$$

$$\mathcal{W}_{t+1} = \mathcal{W}_t - \mu_t \cdot \mathbf{H}_t^{-1} \cdot \nabla E(\mathbf{X}, \mathcal{W}_t) \tag{4c}$$

where W_t is the network weight, H is the Hessian matrix, and $\nabla E(\mathbf{X}, \mathcal{W}_t)$ is the gradient of the error at step t. Regarding this, as the computation requirements are high, the optimization provided by the BFGS algorithm [158] is used, providing an estimation of the Hessian matrix changes

$$\mathcal{W}_{t+1} = \mathcal{W}_t - \mu_t \cdot \mathbf{G}_t \nabla E(\mathbf{X}, \mathcal{W}_t) \tag{5}$$

with $\mathbf{G}_t$ being the inverse Hessian approximation matrix. Since $\mathbf{G}$ is the inverse of the Hessian matrix $\mathbf{H}^{-1}$, and the approximation matrix $\mathbf{G}$ needs to be updated in each step, the BFGS method updates it using the following equation, assuming that $\mathbf{q}_t = \mathbf{H} \cdot \mathbf{p}_t$ and $\mathbf{H}^{-1} \cdot \mathbf{q}_t = \mathbf{p}_t$:

$$\mathbf{G_{t+1}} = \mathbf{G_t} + \frac{\mathbf{p_t} \cdot \mathbf{p_t^T}}{\mathbf{p_t^T} \cdot \mathbf{q_t}} - \mathbf{G_t} \cdot \frac{\mathbf{q_t} \cdot \mathbf{q_t^T}}{\mathbf{q_t^T} \cdot \mathbf{G_t} \cdot \mathbf{q_t}} \cdot \mathbf{G_t}. \tag{6}$$

This strategy is quite appropriate for hyperspectral data sets because the computation is high and the processing requires repetitively reading the original data set. In this way, a distributed cloud computing implementation can make the MLP faster and highly scalable.

Specifically, our distributed implementation reshapes the hyperspectral data from $\mathbf{X} \in \mathbb{R}^{h\times w\times n_{\text{bands}}}$ to $\mathbf{X} \in \mathbb{R}^{n_{pixels}\times n_{\text{bands}}}$. This way, each row or column collects the full representation of a pixel. The master node reads and divides the original data set over P partitions, which are assigned to the corresponding workers in the cluster. The data are stored in each worker as an RDD.

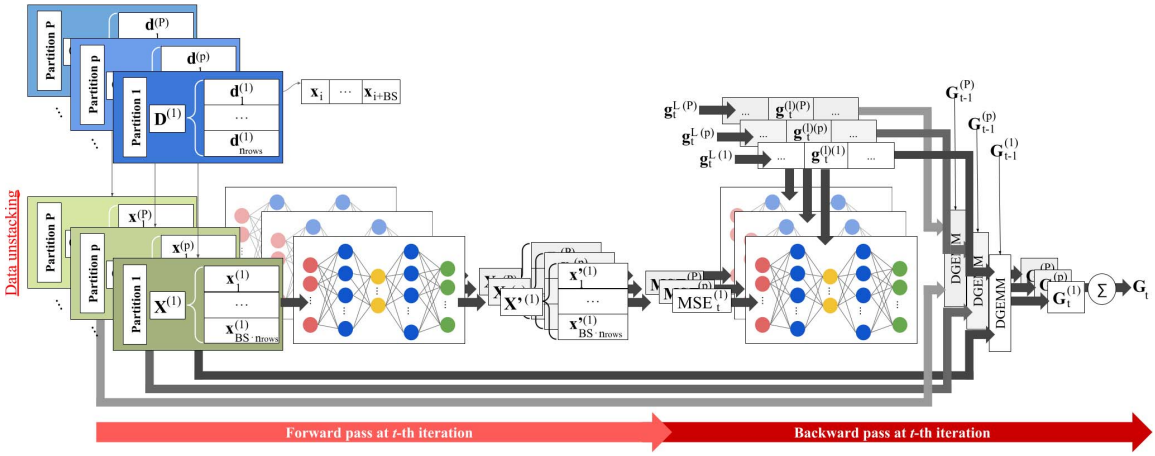


Fig. 11. *Graphical overview of the forward and backward pipelines of our distributed MLP classifier.*

To take advantage of this, we can improve the computation time of the distributed algorithm approaches using a BS implementation. In our implementation, each data partition (row) $\mathbf{r}_j^{(p)}$ is transformed based on the minimum and maximum features of a sample with the minimum and maximum from the column of its data partition as follows:

$$\mathbf{r}_j^{(p)} = \frac{\mathbf{r}_j^{(p)} - \mathbf{r}_{\min}^{(p)}}{\mathbf{r}_{\max}^{(p)} - \mathbf{r}_{\min}^{(p)}} \cdot (\mathbf{x}_{\max} - \mathbf{x}_{\min}) + \mathbf{x}_{\min}. \tag{7}$$

Now, every training iteration can be performed using a forward–backward procedure. An aggregation is done after each step to compute and process the gradients and losses from workers and, hence, return a single gradient and loss. In each forward propagation, each worker forward its corresponding data partition $\mathbf{X}^{(p)}$ through the layers. Then, gradients are computed at the backpropagation step, obtaining, for each partition (p), the $\mathbf{G}_t^{(p)}$ matrix at time step t. Gradients are sent to the master node, and then, the computation of ΔW_t takes place. Assuming that $\mathbf{X}^{(p)} \in (\mathbb{R}^{BS \cdot n_{\text{rows}}) \times n_{\text{bands}}}$ for (p) partitions, (1) is distributed as

$$\mathbf{X}_i^{(l+1),(p)} = \mathcal{H}(\mathbf{X}^{(l),(p)} \cdot \mathbf{W}^{(l)} + b^{(l)}) \tag{8}$$

where $x_i^{(l+1),(p)}$ represents the output matrix neurons of size $(\mathbb{R}^{BS \cdot n_{\text{rows}}) \times n_{\text{neurons}}^l}$ from layer l, $x^{(l),(p)}$ is the input pixel matrix of size $(BS \cdot n_{\text{rows}})$ from the previous layer, $\mathbf{W}^{(l)}$ is the matrix of weights, which connects neurons from previous layers with the actual one, and $\mathcal{H}$ is the activation function (e.g., the ReLU).

Once the forward step is completed in every partition $\mathbf{X}^{(p)}$, the error loss is computed by every worker. The final error is calculated by the master as the mean of all the errors provided by the slaves. Then, the partition error is backpropagated to calculate the $\mathbf{G}_t^{(p)}$ gradient at each time step t. The gradients of each partition are computed using a parallel DGEMM, implemented in BLAS

$$C = \alpha * \mathbf{X}^{(p)} * g_t^{L(p)} + \beta * \mathbf{G}_{t-1}^{(p)} \tag{9}$$

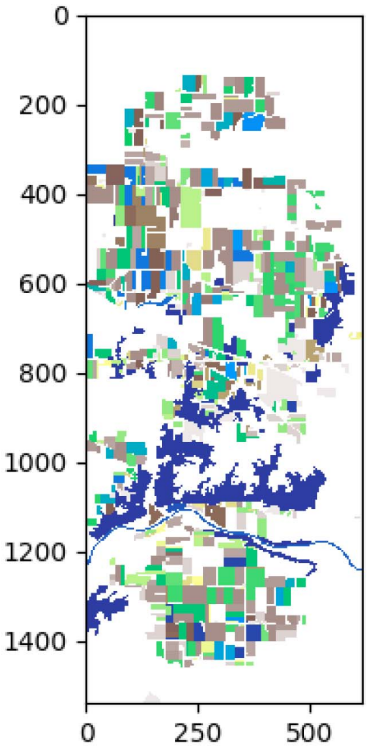
where α and β are regularization parameters set to $1/n_{\text{bands}}$ and 1, respectively, $g_t^{L(p)}$ is a matrix representing the neuron impact per layer $L = [l_1, l_2, \ldots, l_n]$, and $\mathbf{G}_{t-1}^{(p)}$ denotes the previous gradient matrix values. The variable p, as indicated previously, represents the partition of the data. In the end, all partition gradients $\mathbf{G}_t^{(p)}$ are summed to obtain the global gradient matrix $\mathbf{G}_t$ at the time step t.

Fig. 11 provides a graphical description of the distributed forward and backward pipelines of our distributed MLP during the training stage (considering iteration t). This is conducted after unstacking the hyperspectral samples in each distributed data partition, where each one is allocated to a different worker node.

B. Hyperspectral Data Set

To evaluate the performance of our cloud implementation of the MLP classifier, several experiments have been conducted over the BIP data set.[41] It was gathered by the AVIRIS sensor [2] during a flight campaign over the agricultural IP test site in Northwestern Indiana. The scene was collected at the beginning of the 1992 growing season and comprises several regular patches of different crops coupled with irregular forest and grass zones. The data cube comprises 1417 × 617 pixels, with a ground sampling distance of 20 mpp. Furthermore, each pixel comprises 220 channels recorded over a spectral range of 400–500 nm, with a nominal spectral resolution of 10 nm. However, 20 bands [0–9, 210–219] were removed in order to avoid null, noisy, and water absorption bands, keeping the remaining 200 bands for experimental purposes.

[41] https://engineering.purdue.edu/~biehl/MultiSpec/hyperspectral.html



Color	Land cover type	Samples	Color	Land cover type	Samples
	Background	1310047		BareSoil	57
	Buildings	17195		Concrete/Asphalt	69
	Corn	17783		Corn?	158
	Corn-EW	514		Corn-NS	2356
	Corn-CleanTill	12404		Corn-CleanTill-EW	26486
	Corn-CleanTill-NS	39678		Corn-CleanTill-NS-Irrigated	800
	Corn-CleanTilled-NS?	1728		Corn-MinTill	1049
	Corn-MinTill-EW	5629		Corn-MinTill-NS	8862
	Corn-NoTill	4381		Corn-NoTill-EW	1206
	Corn-NoTill-NS	5685		Fescue	114
	Grass	1147		Grass/Trees	2331
	Grass/Pasture-mowed	19		Grass/Pasture	73
	Grass-runway	37		Hay	1128
	Hay?	2185		Hay-Alfalfa	2258
	Lake	224		NotCropped	1940
	Oats	1742		Oats?	335
	Orchard	39		Pasture	10386
	pond	102		Soybeans	9391
	Soybeans?	894		Soybeans-NS	1110
	Soybeans-CleanTill	5074		Soybeans-CleanTill?	2726
	Soybeans-CleanTill-EW	11802		Soybeans-CleanTill-NS	10387
	Soybeans-CleanTill-Drilled	2242		Soybeans-CleanTill-Weedy	543
	Soybeans-Drilled	15118		Soybeans-MinTill	2667
	Soybeans-MinTill-EW	1832		Soybeans-MinTill-Drilled	8098
	Soybeans-MinTill-NS	4953		Soybeans-NoTill	2157
	Soybeans-NoTill-EW	2533		Soybeans-NoTill-NS	929
	Soybeans-NoTill-Drilled	8731		Swampy Area	583
	River	3110		Trees?	580
	Wheat	4979		Woods	63562
	Woods?	144			
		Total samples 1644292			

Fig. 12. ***Available labeled samples (and their distribution) in the AVIRIS BIP data set considered in experiments.***

The complexity of this challenging image is quite remarkable, as the pixels are very mixed due to the low spatial resolution, while the available ground truth is composed of 58 different and highly unbalanced land-cover classes, covering only 20.33% of the samples. The size of the data set exceeds 1 GB. Fig. 12 shows the available labeled samples per class and their distribution in the scene.

C. Platform Configuration

The designed experiments have been conducted over an OpenStack-based cloud infrastructure, which has been implemented onto a hardware platform composed of two × Intel Xeon CPUs E5-2650v2 @2.60 GHz with eight cores (16 way multitask processing), 16-GB RAM, and 600 GB of HDD SAS 10k. In this sense, within the cloud environment, nine VMs (one master instance and eight slave instances) have been launched. Each VM runs Ubuntu 20.04 as OS, with Spark 3.0.1 and Java 9.0.4 serving as running platforms. Furthermore, the Spark framework provides the distributed *MLlib* library, which is used to support the implementation of our cloud-based MLP classifier.[42]

[42]The source code used in this experimentation is currently available on https://github.com/mhaut/cloud-dnn-HSI

D. Experimental Discussion

During the experimentation, the computational load of a fully connected deep network has been distributed in order to evaluate the performance of the cloud infrastructure when dealing with hyperspectral remote sensing image classification. In particular, a deep MLP has been designed to explore the impact of its computational burden over the cloud environment by involving all model parameters with all input elements in the computation of the matrix operations described by (1).

Table 4 describes the architecture of the implemented MLP, specifying the number of layers and the number of neurons comprised by each layer. It is noteworthy that a fully connected neural model entails $\sum_i N_i N_{i+1}$ trainable parameters (where N_i represents the number of nodes at the ith layer with $i = [1, L-1]$, and L is the number of layers), which, in turn, involves approximately $\sum_i 2N_i N_{i+1}$ FLOPs. In particular, the implemented MLP comprises 78 624 trainable parameters, which implies at least 157 248 FLOPs. Furthermore, in the calculation of

Table 4 Configuration of the Implemented MLP Classifier (Number of Neurons per Layer)

Input	Hidden 1	Hidden 2	Hidden 3	Output
200	144	144	144	58

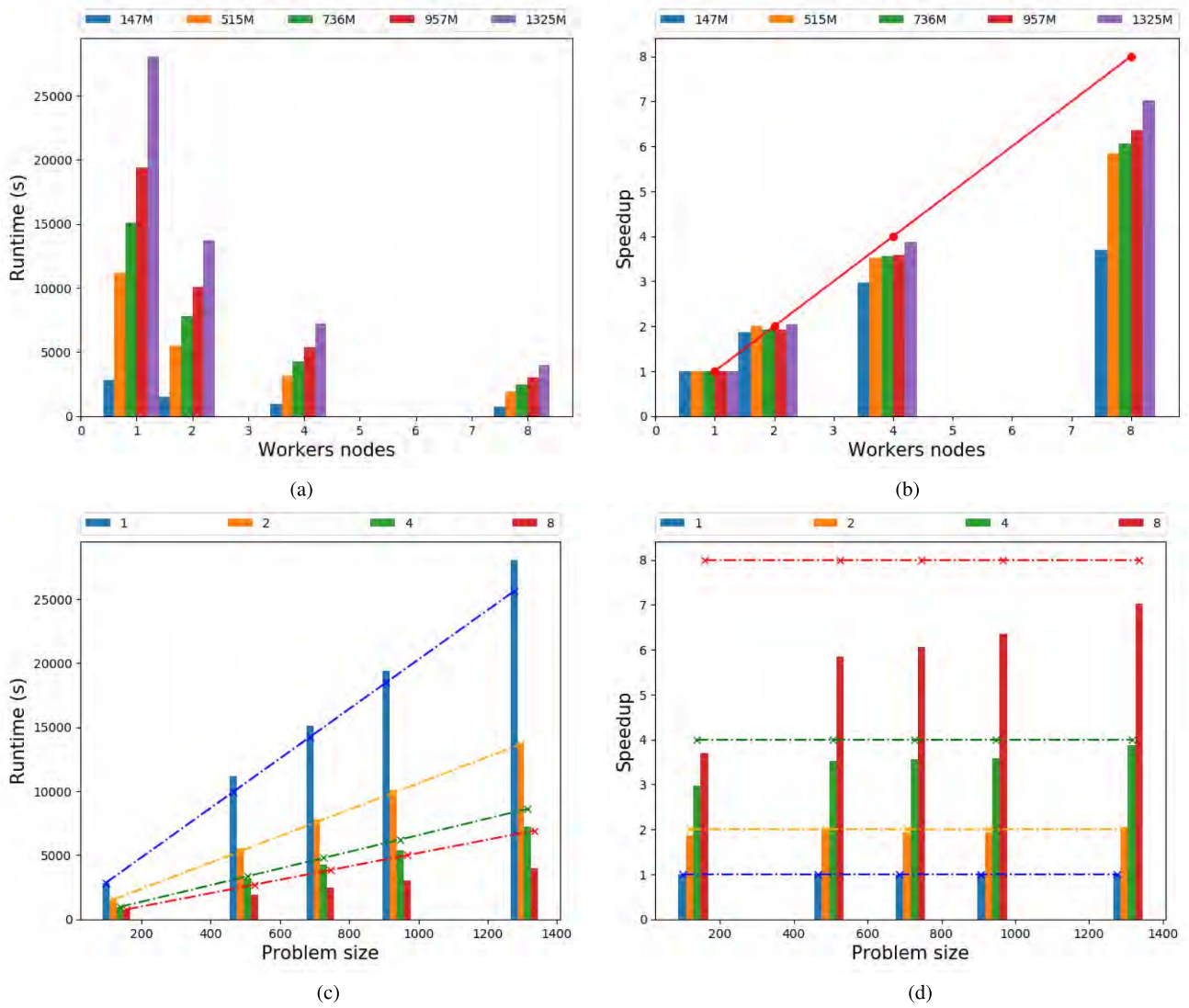


Fig. 13. ***Distributed MLP performance. (a) and (b) Runtimes and speedups as the number of workers increases, considering different amounts of training samples for each one. (c) and (d) Obtained runtimes and speedups as the problem size increases, evaluating the theoretical and obtained behavior for a different number of workers. The different problem sizes result from the selection of different amounts of training data. Depicted lines provide theoretical speedups and expected runtime measurements, while bars provide the actual values obtained.***

FLOPs, the number of samples processed by the model must be taken into account.

In addition, two approaches have been followed to assess the obtained performance in terms of runtimes and speedup. Indeed, experiments have been conducted considering both different numbers of workers and different amounts of training data, where the first approach analyses the obtained runtimes and speedup by focusing on the number of workers, while the second approach evaluates the scalability of the cloud environment by focusing on the amount of data. For each experiment, we conduct five Monte Carlo runs and report the average results.

1) Approach Based on the Cloud Environment Size: The first approach distributes the MLP over the cloud infrastructure to measure the overall performance improvement of the system by evaluating its potential with a different number of workers. In this regard, a cloud environment with one, two, four, and eight working nodes has been launched. Moreover, for each configuration, different amounts of training data have been considered. Particularly, 10%, 35%, 50%, 65%, and 90% of the available labeled data have been randomly selected from the BIP data set to learn the 78 624 trainable parameters comprised by the MLP.

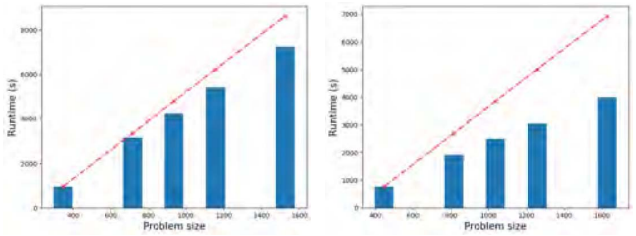
Fig. 13(a) shows a graphical representation of the obtained results in terms of runtime. As we can observe, for each configuration of the cloud infrastructure, five measurements have been collected, which corresponds to the number of training samples. In this regard, focusing on each configuration, the obtained runtimes increase as more data are included during the training stage. This is

a reasonable and expected behavior since, with the same resources, an increase in the problem size brings a corresponding increase in the runtime. However, comparing the runtimes among the different configurations, with only one worker, the execution time soars, easily exceeding 20 000 s with 90% of training samples (purple bar). On the contrary, as the number of workers increases, the observed runtimes decrease significantly. In particular, with eight workers, the highest runtime with 90% of the training samples never exceeds 4000 s.

These results have a clear impact on the speedup of the MLP model, which is increased as more workers are launched into the cloud environment. Fig. 13(b) shows a graphical description of the obtained speedups. Once again, for each configuration, five measurements are depicted corresponding with different training percentages. In this regard, the cloud infrastructure with one worker node has been considered as the baseline configuration, which exhibits a speedup of 1 for all training sizes. Therefore, the speedups of the following configurations have been consequently obtained. It should be noted that the improvement in speedup is not exactly the same as the theoretical speedup marked by the red line in Fig. 13(b), as communication and scheduling times inevitably affect the system performance. However, the data volume is high enough to take full advantage of the cloud environment, without the communication bottleneck preventing a good performance result in terms of runtimes. This is clearly evident in every configuration of the cloud environment. For instance, focusing on configurations with two and four workers, the speedups obtained with different training percentages are quite similar, suggesting that, already, with 10%–50% of training data, computational resources are being optimally exploited. However, with eight working nodes, the speedup is significantly higher when more data are processed, as computational resources are much larger, thus providing more room to exploit a bigger amount of data (i.e., to process larger data sets).

2) Approach Based on Problem Size: As mentioned above, the second approach evaluates the behavior of the implemented cloud solution by placing the focus on the problem size, i.e., by considering different training set sizes. In this regard, the experiment attempts to measure the scalability of each of the cloud environment configurations adopted to solve the problem (with one, two, four, and eight nodes) when different amounts of labeled samples are considered at the training stage. As in the previous experiment, 10%, 35%, 50%, 65%, and 90% of the available labeled samples have been randomly selected to comprise the training sets, resulting in different data sizes (in MBs), which are indicated on the x-axis of the plots reported in Fig. 13(c) and (d).

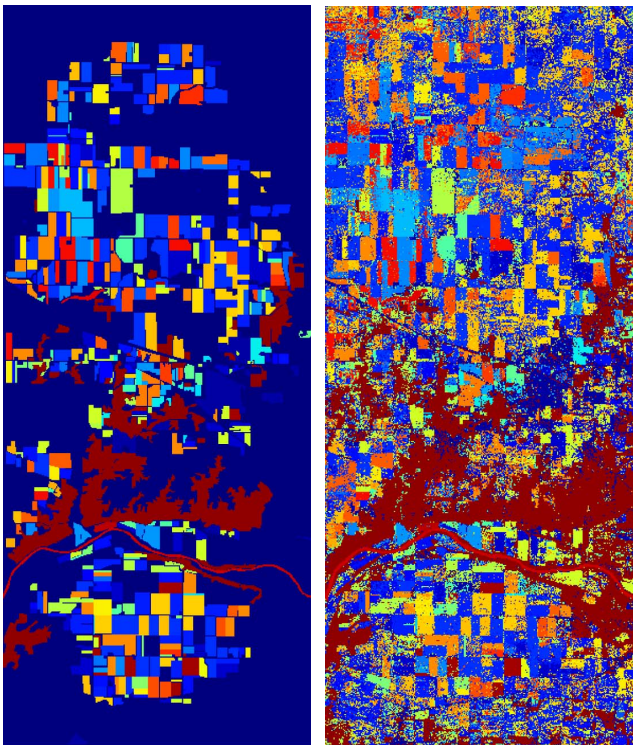
Fig. 13(c) provides the obtained results in terms of runtimes. Following the previous results, for each amount of training data, runtimes decrease as more workers are launched into the cloud environment. In this sense,



(a) (b)

Fig. 14. ***Runtime details of the distributed MLP as the problem size grows, with (a) four and (b) eight working nodes. The different problem sizes result from the selection of different amounts of training data. Red lines provide expected measurements, while blue bars provide the actual measured values.***

the one-worker configuration is the slowest one, with runtimes that are more than seven times longer than the ones obtained by the eight-worker configuration for the case with the most training data. In this sense, if we connect the top of each bar, it is particularly interesting to observe the slope of the line, where the one corresponding to the one-worker configuration is the steepest. It even exceeds the theoretical line, which makes it the worst configuration. On the contrary, the four- and eight-worker configurations scale remarkably well. Moreover, they even give better times than theoretically expected. Fig. 14(a) and (b)



(a) (b)

Fig. 15. ***Classification map obtained by the cloud-based MLP for the BIP data set using 50% of the available labeled data for training. (a) Ground truth. (b) Predicted labels.***

Table 5 OA Comparison

Nodes	10%	35%	50%	65%	90%
1	63.65±0.36	77.57±0.62	81.27±0.29	83.04±0.21	85.16±0.34
2	63.50±0.28	77.67±0.63	80.99±0.14	83.25±0.26	85.39±0.08
4	63.62±0.48	77.56±0.12	81.15±0.38	83.22±0.33	85.80±0.09
8	63.24±0.51	77.76±0.34	81.25±0.18	83.44±0.11	85.40±0.36
Parallel (PyTorch)	61.05 ±1.81	77.45 ±0.49	80.50 ±0.34	82.24 ±0.38	84.37 ±0.33

provides the runtime details for these configurations. As it can be observed in Fig. 14(a), although the runtime increases (as expected), the curve is not entirely linear, which implies that the distribution model is more optimal when there are more data to distribute. Moreover, the distance between the obtained runtimes and the theoretical ones (which are highlighted as a red line) increases as more data are processed. Particularly, the theoretical runtimes times are 1.18 times higher than the obtained ones. This proves that the model scales appropriately with the size of the problem. This is clearly visible in Fig. 14(b), where theoretical runtimes are 1.59 times higher than those currently obtained.

Finally, Fig. 13(d) provides the obtained speedups regarding the problem size. Once more, the one-worker configuration has been considered as the baseline where, for each size of the training set, its speedup is set to 1. Therefore, the speedups of the two-, four-, and eight-worker configurations have been consequently obtained. As in the previous plots, the theoretical speedups have been marked as dotted lines for each configuration. In this sense, the speedup exhibited by the two-worker configuration is quite close to the theoretical one, while, for the four- and eight-worker configurations, their speedups improve as the size of the processed data grows. This is particularly evident when eight working nodes are launched into the cloud environment. These results indicate that the computational resources provided by the eight-worker configuration exhibit great scalability, with a great potential to process bigger remote sensing data sets in order to optimize the use of the cloud environment capacities.

3) Accuracy Evaluation: Finally, the reliability of the classification results has been measured in terms of overall accuracy (OA). In this sense, Table 5 provides the OAs that have been obtained after training the deep MLP with 10%, 35%, 50%, 65%, and 90% of randomly selected samples. Furthermore, for the cloud-distributed MLP, configurations with one, two, four, and eight working nodes have been considered. These results have been compared with a parallel implementation based on the PyTorch framework. As we can observe, the obtained OAs improve as the MLP network is trained with more samples. Moreover, after comparing the different implementations, we can see that the obtained results are quite similar. Fig. 15 provides a classification map that has been obtained by training the cloud-based MLP with 50% of the available labeled samples. In this regard, the cloud solution not only provides an efficient way to distribute the storage and computation load but also reaches good performance in terms of accuracy.

To conclude this section, we emphasize that the results that have been obtained with the MLP are perfectly extrapolable to other deep networks, such as CNNs. Indeed, the cloud environment can run both architectures in a distributed manner (as it is not specialized hardware), reaching impressive performance in image analysis with CNNs as well [159].

VII. CONCLUSION AND FUTURE LINES

In this article, we have presented a comprehensive review of recent efforts in parallel and distributed processing of remotely sensed images, with a particular emphasis on DL-based approaches and their cloud implementation. Our review reflects the growing importance of using cloud computing techniques for distributed processing of remote sensing images, which is of great importance due to the current availability of open big remote sensing data repositories. Our review also summarized the processing tools and techniques that have been used in different remote sensing applications, which is believed to provide a useful guideline for new users that wish to develop computationally efficient techniques in this field.

We provide a case study illustrating the results obtained by a DL algorithm (implemented in the cloud) when processing a big hyperspectral image. Since hyperspectral data are characterized by their large size and complex processing and storage requirements, we believe that the results provided in our case study offer a good perspective on the possibilities of implementing DL algorithms in the cloud for addressing the processing challenges involved in the extraction of information from remotely sensed images. In the future, we will expand our study by considering the inclusion and optimization of specific accelerators (such as GPUs) in the cloud environment for DL-based remote sensing data processing and interpretation. ■

REFERENCES

[1] B. Zhang *et al.*, "Remotely sensed big data: Evolution in model development for information extraction [point of view]," *Proc. IEEE*, vol. 107, no. 12, pp. 2294–2301, Dec. 2019.

[2] J. Li, J. A. Benediktsson, B. Zhang, T. Yang, and A. Plaza, "Spatial technology and social media in remote sensing: A survey," *Proc. IEEE*, vol. 105, no. 10, pp. 1855–1864, Oct. 2017.

[3] C.-I. Chang, *Hyperspectral Imaging: Techniques for Spectral Detection and Classification*. New York, NY, USA: Kluwer, 2003.

[4] A. Plaza, J. Plaza, A. Paz, and S. Sanchez, "Parallel

hyperspectral image and signal processing [applications corner]," *IEEE Signal Process. Mag.*, vol. 28, no. 3, pp. 119–126, May 2011.
[5] R. O. Green *et al.*, "Imaging spectroscopy and the airborne visible/infrared imaging spectrometer (AVIRIS)," *Remote Sens. Environ.*, vol. 65, no. 3, pp. 227–248, Sep. 1998.
[6] M. Chi, A. Plaza, J. A. Benediktsson, Z. Sun, J. Shen, and Y. Zhu, "Big data for remote sensing: Challenges and opportunities," *Proc. IEEE*, vol. 104, no. 11, pp. 2207–2219, Nov. 2016.
[7] Y. M. L. Wang and J. Yang, *Cloud Computing in Remote Sensing*. Boca Raton, FL, USA: CRC Press, 2019.
[8] C.-I. C. A. Plaza, *High Performance Computing in Remote Sensing*. Boca Raton, FL, USA: CRC Press, 2006.
[9] J. Behnke, T. H. Watts, B. Kobler, D. Lowe, S. Fox, and R. Meyer, "EOSDIS petabyte archives: Tenth anniversary," in *Proc. 22nd IEEE/13th NASA Goddard Conf. Mass Storage Syst. Technol. (MSST)*, Apr. 2005, pp. 81–93.
[10] B. Ryan and D. Cripe, "The Group on Earth Observations (GEO) through 2025," in *Proc. 40th COSPAR Sci. Assembly*, vol. 40, Jan. 2014, pp. 1–14.
[11] P. M. Mell and T. Grance, "The NIST definition of cloud computing," Nat. Inst. Standards Technol., Gaithersburg, MD, USA, Tech. Rep. Sp 800-145, 2011.
[12] J. Plaza, R. Pérez, A. Plaza, P. Martínez, and D. Valencia, "Parallel morphological/neural processing of hyperspectral images using heterogeneous and homogeneous platforms," *Cluster Comput.*, vol. 11, no. 1, pp. 17–32, Mar. 2008.
[13] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016. [Online]. Available: http://www.deeplearningbook.org
[14] L. Zhang, L. Zhang, and B. Du, "Deep learning for remote sensing data: A technical tutorial on the state of the art," *IEEE Geosci. Remote Sens. Mag.*, vol. 4, no. 2, pp. 22–40, Jun. 2016.
[15] L. Ma, Y. Liu, X. Zhang, Y. Ye, G. Yin, and B. A. Johnson, "Deep learning in remote sensing applications: A meta-analysis and review," *ISPRS J. Photogramm. Remote Sens.*, vol. 152, pp. 166–177, Jun. 2019.
[16] X. X. Zhu *et al.*, "Deep learning in remote sensing: A comprehensive review and list of resources," *IEEE Geosci. Remote Sens. Mag.*, vol. 5, no. 4, pp. 8–36, Dec. 2017.
[17] M. E. Paoletti, J. M. Haut, J. Plaza, and A. Plaza, "Deep learning classifiers for hyperspectral imaging: A review," *ISPRS J. Photogramm. Remote Sens.*, vol. 158, pp. 279–317, Dec. 2019.
[18] A. Plaza, Q. Du, Y.-L. Chang, and R. L. King, "High performance computing for hyperspectral remote sensing," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 4, no. 3, pp. 528–544, Sep. 2011.
[19] C. A. Lee, S. D. Gasster, A. Plaza, C.-I. Chang, and B. Huang, "Recent developments in high performance computing for remote sensing: A review," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 4, no. 3, pp. 508–527, Sep. 2011.
[20] A. F. H. Goetz, "Three decades of hyperspectral remote sensing of the Earth: A personal view," *Remote Sens. Environ.*, vol. 113, pp. S5–S16, Sep. 2009.
[21] N. Yokoya, C. Grohnfeldt, and J. Chanussot, "Hyperspectral and multispectral data fusion: A comparative review of the recent literature," *IEEE Geosci. Remote Sens. Mag.*, vol. 5, no. 2, pp. 29–56, Jun. 2017.
[22] C. Liu, J. Shang, P. W. Vachon, and H. McNairn, "Multiyear crop monitoring using polarimetric RADARSAT-2 data," *IEEE Trans. Geosci. Remote Sens.*, vol. 51, no. 4, pp. 2227–2240, Apr. 2013.
[23] A. Rosenqvist, M. Shimada, N. Ito, and M. Watanabe, "ALOS PALSAR: A pathfinder mission for global-scale monitoring of the environment," *IEEE Trans. Geosci. Remote Sens.*, vol. 45, no. 11, pp. 3307–3316, Nov. 2007.
[24] M. Moghaddam *et al.*, "Airborne microwave observatory of subcanopy and subsurface (AirMOSS) Earth venture suborbital mission overview," in *Proc. AGUFM*, 2015, pp. B53A–0537.
[25] W. Pitz and D. Miller, "The TerraSAR-X satellite," *IEEE Trans. Geosci. Remote Sens.*, vol. 48, no. 2, pp. 615–622, Feb. 2010.
[26] D. Geudtner, R. Torres, P. Snoeij, M. Davidson, and B. Rommen, "Sentinel-1 system capabilities and applications," in *Proc. IEEE Geosci. Remote Sens. Symp.*, Jul. 2014, pp. 1457–1460.
[27] D. Yi, J. P. Harbeck, S. S. Manizade, N. T. Kurtz, M. Studinger, and M. Hofton, "Arctic sea ice freeboard retrieval with waveform characteristics for NASA's airborne topographic mapper (ATM) and land, vegetation, and ice sensor (LVIS)," *IEEE Trans. Geosci. Remote Sens.*, vol. 53, no. 3, pp. 1403–1410, Mar. 2015.
[28] N. Levin, K. Johansen, J. M. Hacker, and S. Phinn, "A new source for high spatial resolution night time images—The EROS-B commercial satellite," *Remote Sens. Environ.*, vol. 149, pp. 1–12, Jun. 2014.
[29] Y. Chen, R. Fan, M. Bilal, X. Yang, J. Wang, and W. Li, "Multilevel cloud detection for high-resolution remote sensing imagery using multiple convolutional neural networks," *ISPRS Int. J. Geo-Inf.*, vol. 7, no. 5, p. 181, May 2018.
[30] G. Dial, H. Bowen, F. Gerlach, J. Grodecki, and R. Oleszczuk, "IKONOS satellite, imagery, and products," *Remote Sens. Environ.*, vol. 88, nos. 1–2, pp. 23–36, Nov. 2003.
[31] M. Gašparović, L. Rumora, M. Miler, and D. Medak, "Effect of fusing Sentinel-2 and WorldView-4 imagery on the various vegetation indices," *Proc. SPIE*, vol. 13, no. 3, Jul. 2019, Art. no. 036503.
[32] M. Drusch *et al.*, "Sentinel-2: ESA's optical high-resolution mission for GMES operational services," *Remote Sens. Environ.*, vol. 120, pp. 25–36, May 2012.
[33] C. Donlon *et al.*, "The global monitoring for environment and security (GMES) Sentinel-3 mission," *Remote Sens. Environ.*, vol. 120, pp. 37–57, May 2012.
[34] D. P. Roy *et al.*, "Landsat-8: Science and product vision for terrestrial global change research," *Remote Sens. Environ.*, vol. 145, pp. 154–172, Apr. 2014.
[35] A. Savtchenko *et al.*, "Terra and Aqua MODIS products available from NASA GES DAAC," *Adv. Space Res.*, vol. 34, no. 4, pp. 710–714, Jan. 2004.
[36] M. A. Cutter, D. R. Lobb, and R. A. Cockshott, "Compact high resolution imaging spectrometer (CHRIS)," *Acta Astronautica*, vol. 46, nos. 2–6, pp. 263–268, Jan. 2000.
[37] J. W. Chapman *et al.*, "Spectral and radiometric calibration of the next generation airborne visible infrared spectrometer (AVIRIS-NG)," *Remote Sens.*, vol. 11, no. 18, p. 2129, Sep. 2019.
[38] B. Kunkel, F. Blechinger, R. Lutz, R. Doerffer, H. van der Piepen, and M. Schroder, "ROSIS (Reflective Optics System Imaging Spectrometer)—A candidate instrument for polar platform missions," in *Optoelectronic Technologies for Remote Sensing From Space*, vol. 0868, J. Seeley and S. Bowyer, Eds. Bellingham, WA, USA: SPIE, 1988, p. 8.
[39] S. K. Babey and C. D. Anger, "Compact airborne spectrographic imager (CASI): A progress review," in *Imaging Spectrometry of the Terrestrial Environment*, vol. 1937, G. Vane, Ed. Bellingham, WA, USA: SPIE, 1993, pp. 152–163, doi: 10.1117/12.157052.
[40] L. J. Rickard, R. W. Basedow, E. F. Zalewski, P. R. Silverglate, and M. Landers, "HYDICE: An airborne system for hyperspectral imaging," *Proc. SPIE*, vol. 1937, pp. 173–180, Sep. 1993.
[41] T. Cocks, R. Jenssen, A. Stewart, I. Wilson, and T. Shields, "The HyMap airborne hyperspectral sensor: The system, calibration and performance," in *Proc. 1st EARSeL Workshop Imag. Spectrosc.*, 1998, pp. 37–42.
[42] P. Mouroulis *et al.*, "Portable remote imaging spectrometer coastal ocean sensor: Design, characteristics, and first flight results," *Appl. Opt.*, vol. 53, no. 7, pp. 1363–1380, 2014.
[43] L. Guanter *et al.*, "The EnMAP spaceborne imaging spectroscopy mission for Earth observation," *Remote Sens.*, vol. 7, no. 7, pp. 8830–8857, Jul. 2015.
[44] N. Yokoya and A. Iwasaki, "Hyperspectral and multispectral data fusion mission on hyperspectral imager suite (HISUI)," in *Proc. IEEE Int. Geosci. Remote Sens. Symp. (IGARSS)*, Jul. 2013, pp. 4086–4089.
[45] A. Eckardt *et al.*, "DESIS (DLR Earth sensing imaging spectrometer for the ISS-MUSES platform)," in *Proc. IEEE Int. Geosci. Remote Sens. Symp. (IGARSS)*, Jul. 2015, pp. 1457–1459.
[46] J. S. Pearlman, P. S. Barry, C. C. Segal, J. Shepanski, D. Beiso, and S. L. Carman, "Hyperion, a space-based imaging spectrometer," *IEEE Trans. Geosci. Remote Sens.*, vol. 41, no. 6, pp. 1160–1173, Jun. 2003.
[47] C. Galeazzi, A. Sacchetti, A. Cisbani, and G. Babini, "The PRISMA program," in *Proc. IEEE Int. Geosci. Remote Sens. Symp. (IGARSS)*, vol. 4, Jul. 2008, pp. IV–105.
[48] T. Feingersh and E. Ben Dor, "SHALOM—A commercial hyperspectral space mission," in *Optical Payloads for Space Missions*. Hoboken, NJ, USA: Wiley, 2015, ch. 11, pp. 247–263. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/9781118945179.ch11, doi: 10.1002/9781118945179.ch11.
[49] G. A. Swayze, "Effects of spectrometer band pass, sampling, and signal-to-noise ratio on spectral identification using the Tetracorder algorithm," *J. Geophys. Res.*, vol. 108, no. E9, pp. 5105–5135, 2003.
[50] A. F. H. Goetz, G. Vane, J. E. Solomon, and B. N. Rock, "Imaging spectrometry for Earth remote sensing," *Science*, vol. 228, no. 4704, pp. 1147–1153, 1985.
[51] S. K. Seelan, S. Laguette, G. M. Casady, and G. A. Seielstad, "Remote sensing applications for precision agriculture: A learning community approach," *Remote Sens. Environ.*, vol. 88, nos. 1–2, pp. 157–169, Nov. 2003.
[52] H. M. Pham, Y. Yamaguchi, and T. Q. Bui, "A case study on the relation between city planning and urban growth using remote sensing and spatial metrics," *Landscape Urban Planning*, vol. 100, no. 3, pp. 223–230, Apr. 2011.
[53] S. A. Azzouzi, A. Vidal-Pantaleoni, and H. A. Bentounes, "Desertification monitoring in Biskra, Algeria, with landsat imagery by means of supervised classification and change detection methods," *IEEE Access*, vol. 5, pp. 9065–9072, 2017.
[54] X. Ceamanos and S. Valero, "Processing hyperspectral images," in *Optical Remote Sensing of Land Surface*. Amsterdam, The Netherlands: Elsevier, 2016, pp. 163–200.
[55] A. Maffei, J. M. Haut, M. E. Paoletti, J. Plaza, L. Bruzzone, and A. Plaza, "A single model CNN for hyperspectral image denoising," *IEEE Trans. Geosci. Remote Sens.*, vol. 58, no. 4, pp. 2516–2529, Apr. 2020.
[56] G. Cheng, X. Xie, J. Han, L. Guo, and G.-S. Xia, "Remote sensing image scene classification meets deep learning: Challenges, methods, benchmarks, and opportunities," 2020, *arXiv:2005.01094*. [Online]. Available: http://arxiv.org/abs/2005.01094
[57] M. E. Paoletti, J. M. Haut, J. Plaza, and A. Plaza, "A new deep convolutional neural network for fast hyperspectral image classification," *ISPRS J. Photogramm. Remote Sens.*, vol. 145, pp. 120–147, Nov. 2018.
[58] V. Walter, "Object-based classification of remote sensing data for change detection," *ISPRS J. Photogramm. Remote Sens.*, vol. 58, nos. 3–4, pp. 225–238, Jan. 2004.
[59] K. Nogueira, O. A. B. Penatti, and J. A. dos Santos, "Towards better exploiting convolutional neural networks for remote sensing scene classification," *Pattern Recognit.*, vol. 61, pp. 539–556, Jan. 2017.

[60] J. Plaza, A. Plaza, R. Perez, and P. Martinez, "On the use of small training sets for neural network-based characterization of mixed pixels in remotely sensed hyperspectral images," *Pattern Recognit.*, vol. 42, no. 11, pp. 3032–3045, Nov. 2009.

[61] J. A. G. Jaramago, M. E. Paoletti, J. M. Haut, R. Fernandez-Beltran, A. Plaza, and J. Plaza, "GPU parallel implementation of dual-depth sparse probabilistic latent semantic analysis for hyperspectral unmixing," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 12, no. 9, pp. 3156–3167, Sep. 2019.

[62] G. Camps-Valls, "Machine learning in remote sensing data processing," in *Proc. IEEE Int. Workshop Mach. Learn. Signal Process.*, Sep. 2009, pp. 1–6.

[63] D. J. Lary, A. H. Alavi, A. H. Gandomi, and A. L. Walker, "Machine learning in geosciences and remote sensing," *Geosci. Frontiers*, vol. 7, no. 1, pp. 3–10, 2016.

[64] J. M. Haut, R. Fernandez-Beltran, M. E. Paoletti, J. Plaza, A. Plaza, and F. Pla, "A new deep generative network for unsupervised remote sensing single-image super-resolution," *IEEE Trans. Geosci. Remote Sens.*, vol. 56, no. 11, pp. 6792–6810, Nov. 2018.

[65] D. Tuia and G. Camps-Valls, "Semisupervised remote sensing image classification with cluster kernels," *IEEE Geosci. Remote Sens. Lett.*, vol. 6, no. 2, pp. 224–228, Apr. 2009.

[66] D. Tuia, M. Volpi, L. Copa, M. Kanevski, and J. Munoz-Mari, "A survey of active learning algorithms for supervised remote sensing image classification," *IEEE J. Sel. Topics Signal Process.*, vol. 5, no. 3, pp. 606–617, Jun. 2011.

[67] Y. Li, K. Fu, H. Sun, and X. Sun, "An aircraft detection framework based on reinforcement learning and convolutional neural networks in remote sensing images," *Remote Sens.*, vol. 10, no. 2, p. 243, Feb. 2018.

[68] J. M. Haut, M. Paoletti, J. Plaza, and A. Plaza, "Cloud implementation of the K-means algorithm for hyperspectral image analysis," *J. Supercomput.*, vol. 73, no. 1, pp. 514–529, Jan. 2017.

[69] E. Blanzieri and F. Melgani, "Nearest neighbor classification of remote sensing images with the maximal margin principle," *IEEE Trans. Geosci. Remote Sens.*, vol. 46, no. 6, pp. 1804–1811, Jun. 2008.

[70] S. Delalieux, B. Somers, B. Haest, T. Spanhove, J. Vanden Borre, and C. A. Mücher, "Heathland conservation status mapping through integration of hyperspectral mixture analysis and decision tree classifiers," *Remote Sens. Environ.*, vol. 126, pp. 222–231, Nov. 2012.

[71] K. Y. Peerbhay, O. Mutanga, and R. Ismail, "Random forests unsupervised classification: The detection and mapping of solanum mauritianum infestations in plantation forestry using hyperspectral data," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 8, no. 6, pp. 3107–3122, Jun. 2015.

[72] J. M. Haut and M. E. Paoletti, "Cloud implementation of multinomial logistic regression for UAV hyperspectral images," *IEEE J. Miniaturization Air Space Syst.*, vol. 1, no. 3, pp. 163–171, Dec. 2020.

[73] J. Ju, E. D. Kolaczyk, and S. Gopal, "Gaussian mixture discriminant analysis and sub-pixel land cover characterization in remote sensing," *Remote Sens. Environ.*, vol. 84, no. 4, pp. 550–560, Apr. 2003.

[74] J. Yang, Z. Ye, X. Zhang, W. Liu, and H. Jin, "Attribute weighted naive Bayes for remote sensing image classification based on cuckoo search algorithm," in *Proc. Int. Conf. Secur., Pattern Anal., Cybern. (SPAC)*, Dec. 2017, pp. 169–174.

[75] P. B. C. Leite, R. Q. Feitosa, A. R. Formaggio, G. A. O. P. da Costa, K. Pakzad, and I. D. Sanches, "Hidden Markov models for crop recognition in remote sensing image sequences," *Pattern Recognit. Lett.*, vol. 32, no. 1, pp. 19–26, Jan. 2011.

[76] M. E. Paoletti, J. M. Haut, X. Tao, J. P. Miguel, and A. Plaza, "A new GPU implementation of support vector machines for fast hyperspectral image classification," *Remote Sens.*, vol. 12, no. 8, p. 1257, Apr. 2020.

[77] S. Yang, Q. Feng, T. Liang, B. Liu, W. Zhang, and H. Xie, "Modeling grassland above-ground biomass based on artificial neural network and remote sensing in the three-river headwaters region," *Remote Sens. Environ.*, vol. 204, pp. 448–455, Jan. 2018.

[78] Y. Chen, Z. Lin, X. Zhao, G. Wang, and Y. Gu, "Deep learning-based classification of hyperspectral data," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 7, no. 6, pp. 2094–2107, Jun. 2014.

[79] M. E. Paoletti, J. M. Haut, J. Plaza, and A. Plaza, "Scalable recurrent neural network for hyperspectral image classification," *J. Supercomput.*, vol. 76, pp. 8866–8882, Feb. 2020.

[80] M. E. Paoletti, J. M. Haut, R. Fernandez-Beltran, J. Plaza, A. J. Plaza, and F. Pla, "Deep pyramidal residual networks for spectral–spatial hyperspectral image classification," *IEEE Trans. Geosci. Remote Sens.*, vol. 57, no. 2, pp. 740–754, Feb. 2019.

[81] M. E. Paoletti *et al.*, "Capsule networks for hyperspectral image classification," *IEEE Trans. Geosci. Remote Sens.*, vol. 57, no. 4, pp. 2145–2160, Apr. 2019.

[82] D. Lunga, J. Gerrand, L. Yang, C. Layton, and R. Stewart, "Apache spark accelerated deep learning inference for large scale satellite image analytics," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 13, pp. 271–283, 2020.

[83] M. Ma *et al.*, "Democratizing production-scale distributed deep learning," 2018, *arXiv:1811.00143*. [Online]. Available: https://arxiv.org/abs/1811.00143

[84] R. Sedona, G. Cavallaro, J. Jitsev, A. Strube, M. Riedel, and J. Benediktsson, "Remote sensing big data classification with high performance distributed deep learning," *Remote Sens.*, vol. 11, no. 24, p. 3056, Dec. 2019. [Online]. Available: https://www.mdpi.com/2072-4292/11/24/3056

[85] Y. Lu, K. Xie, G. Xu, H. Dong, C. Li, and T. Li, "MTFC: A multi-GPU training framework for cube-CNN-based hyperspectral image classification," *IEEE Trans. Emerg. Topics Comput.*, early access, Aug. 17, 2020, doi: 10.1109/TETC.2020.3016978.

[86] M. Aspri, G. Tsagkatakis, and P. Tsakalides, "Distributed training and inference of deep learning models for multi-modal land cover classification," *Remote Sens.*, vol. 12, no. 17, p. 2670, Aug. 2020. [Online]. Available: https://www.mdpi.com/2072-4292/12/17/2670

[87] Y. Li, Z. Wu, J. Wei, A. Plaza, J. Li, and Z. Wei, "Fast principal component analysis for hyperspectral imaging based on cloud computing," in *Proc. IEEE Int. Geosci. Remote Sens. Symp. (IGARSS)*, Milan, Italy, Jul. 2015, pp. 513–516.

[88] Z. Wu, Y. Li, A. Plaza, J. Li, F. Xiao, and Z. Wei, "Parallel and distributed dimensionality reduction of hyperspectral data on cloud computing architectures," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 9, no. 6, pp. 2270–2278, Jun. 2016.

[89] J. Gu, Z. Wu, Y. Li, Y. Chen, Z. Wei, and W. Wang, "Parallel optimization of pixel purity index algorithm for hyperspectral unmixing based on spark," in *Proc. 3rd Int. Conf. Adv. Cloud Big Data*, Oct. 2015, pp. 159–166. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7435468

[90] C. Bielski, S. Gentilini, and M. Pappalardo, "Post-disaster image processing for damage analysis using GENESI-DR, WPS and grid computing," *Remote Sens.*, vol. 3, no. 6, pp. 1234–1250, Jun. 2011. [Online]. Available: https://www.mdpi.com/2072-4292/3/6/1234

[91] J. M. Haut *et al.*, "Cloud deep networks for hyperspectral image analysis," *IEEE Trans. Geosci. Remote Sens.*, vol. 57, no. 12, pp. 9832–9848, Dec. 2019.

[92] Z. Wu *et al.*, "Scheduling-guided automatic processing of massive hyperspectral image classification on cloud computing architectures," *IEEE Trans. Cybern.*, early access, Oct. 29, 2020, doi: 10.1109/TCYB.2020.3026673.

[93] L. Parente, E. Taquary, A. Silva, C. Souza, and L. Ferreira, "Next generation mapping: Combining deep learning, cloud computing, and big remote sensing data," *Remote Sens.*, vol. 11, no. 23, p. 2881, Dec. 2019. [Online]. Available: https://www.mdpi.com/2072-4292/11/23/2881

[94] X. Yao *et al.*, "Enabling the big Earth observation data via cloud computing and DGGS: Opportunities and challenges," *Remote Sens.*, vol. 12, no. 1, p. 62, Dec. 2019. [Online]. Available: https://www.mdpi.com/2072-4292/12/1/62

[95] P. Zheng *et al.*, "A parallel unmixing-based content retrieval system for distributed hyperspectral imagery repository on cloud computing platforms," *Remote Sens.*, vol. 13, no. 2, p. 176, Jan. 2021. [Online]. Available: https://www.mdpi.com/2072-4292/13/2/176

[96] J. M. Bioucas-Dias *et al.*, "Hyperspectral unmixing overview: Geometrical, statistical, and sparse regression-based approaches," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 5, no. 2, pp. 354–379, Apr. 2012.

[97] W. Huang, L. Meng, D. Zhang, and W. Zhang, "In-memory parallel processing of massive remotely sensed data using an Apache spark on Hadoop YARN model," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 10, no. 1, pp. 3–19, Jan. 2017.

[98] V. A. Ayma *et al.*, "On the architecture of a big data classification tool based on a map reduce approach for hyperspectral image analysis," in *Proc. IEEE Int. Geosci. Remote Sens. Symp. (IGARSS)*, Jul. 2015, pp. 1508–1511.

[99] P. Bajcsy, P. Nguyen, A. Vandecreme, and M. Brady, "Spatial computations over terabyte-sized images on Hadoop platforms," in *Proc. IEEE Int. Conf. Big Data (Big Data)*, Oct. 2014, pp. 816–824.

[100] P. N. Happ, R. S. Ferreira, G. A. O. P. Costa, R. Q. Feitosa, C. Bentes, and P. Gamba, "Towards distributed region growing image segmentation based on MapReduce," in *Proc. IEEE Int. Geosci. Remote Sens. Symp. (IGARSS)*, Jul. 2015, pp. 4352–4355. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=7326790

[101] J. Xing and R. Sieber, "Sampling based image splitting in large scale distributed computing of Earth observation data," in *Proc. IEEE Geosci. Remote Sens. Symp.*, Jul. 2014, pp. 1409–1412. [Online]. Available: http://www.scopus.com/inward/record.url?eid=2-s2.0-84911366834&partnerID=tZOtx3y1

[102] X. Pan and S. Zhang, "A remote sensing image cloud processing system based on Hadoop," in *Proc. IEEE 2nd Int. Conf. Cloud Comput. Intell. Syst.*, vol. 1, Oct. 2012, pp. 492–494.

[103] Y. Liu, B. Chen, W. He, and Y. Fang, "Massive image data management using HBase and MapReduce," in *Proc. 21st Int. Conf. Geoinformatics*, Jun. 2013.

[104] R. Rajak, D. Raveendran, M. C. Bh, and S. S. Medasani, "High resolution satellite image processing using Hadoop framework," in *Proc. IEEE Int. Conf. Cloud Comput. Emerg. Markets (CCEM)*, Nov. 2015, pp. 16–21. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7436925

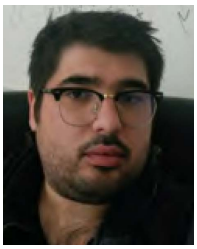
[105] M. M. Rathore, A. Ahmad, A. Paul, and A. Daniel, "Hadoop based real-time big data architecture for remote sensing Earth observatory system," in *Proc. 6th Int. Conf. Comput., Commun. Netw. Technol. (ICCCNT)*, vol. 1, Jul. 2015, pp. 1–7.

[106] W. Nina *et al.*, "A new approach to the massive processing of satellite images," in *Proc. 41st Latin Amer. Comput. Conf. (CLEI)*, Oct. 2015, pp. 1–6.

[107] Y. Zhong, J. Fang, and X. Zhao, "VegaIndexer: A distributed composite index scheme for big

spatio-temporal sensor data on cloud," in *Proc. IEEE Int. Geosci. Remote Sens. Symp. (IGARSS)*, Jul. 2013, pp. 1713–1716.
[108] R. Kune, P. Konugurthi, A. Agarwal, R. R. Chillarige, and R. Buyya, "XHAMI—Extended HDFS and MapReduce interface for image processing applications," in *Proc. IEEE Int. Conf. Cloud Comput. Emerg. Markets (CCEM)*, Nov. 2015, pp. 43–51.
[109] M. T. Patterson *et al.*, "The Matsu wheel: A cloud-based framework for efficient analysis and reanalysis of Earth satellite imagery," in *Proc. IEEE 2nd Int. Conf. Big Data Comput. Service Appl. (BigDataService)*, Mar. 2016, pp. 156–165.
[110] A. Plaza, D. Valencia, J. Plaza, and P. Martinez, "Commodity cluster-based parallel processing of hyperspectral imagery," *J. Parallel Distrib. Comput.*, vol. 66, no. 3, pp. 345–358, 2006.
[111] G. Aloisio and M. Cafaro, "A dynamic Earth observation system," *Parallel Comput.*, vol. 29, no. 10, pp. 1357–1362, Oct. 2003.
[112] D. Gorgan, V. Bacu, T. Stefanut, D. Rodila, and D. Mihon, "Grid based satellite image processing platform for Earth observation application development," in *Proc. IEEE Int. Workshop Intell. Data Acquisition Adv. Comput. Syst., Technol. Appl.*, Sep. 2009, pp. 247–252.
[113] Z. Chen, N. Chen, C. Yang, and L. Di, "Cloud computing enabled Web processing service for Earth observation data processing," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 5, no. 6, pp. 1637–1649, Dec. 2012.
[114] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The Hadoop distributed file system," in *Proc. IEEE 26th Symp. Mass Storage Syst. Technol. (MSST)*, May 2010, pp. 1–10.
[115] V. K. Vavilapalli *et al.*, "Apache Hadoop YARN: Yet another resource negotiator," in *Proc. 4th Annu. Symp. Cloud Comput.*, Oct. 2013, pp. 1–3, doi: 10.1145/2523616.2523633.
[116] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets," *Proc. 2nd USENIX Conf. Hot Topics Cloud Comput. (HotCloud)*, 2010, p. 10.
[117] M. Zaharia, M. Chowdhury, T. Das, and A. Dave, "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing," in *Proc. 9th USENIX Conf. Netw. Syst. Design Implement. (NSDI)*, 2012, p. 2. [Online]. Available: https://www.usenix.org/system/files/conference/nsdi12/nsdi12-final138.pdf
[118] X. Meng *et al.*, "MLlib: Machine learning in Apache spark," 2015, *arXiv:1505.06807*. [Online]. Available: http://arxiv.org/abs/1505.06807
[119] D. Marinescu, *Cloud Computing: Theory and Practice*. Amsterdam, The Netherlands: Elsevier, 2017. [Online]. Available: https://books.google.es/books?id=O9smDwAAQBAJ
[120] M. Armbrust *et al.*, "A view of cloud computing," *Commun. ACM*, vol. 53, no. 4, pp. 50–58, Apr. 2010, doi: 10.1145/1721654.1721672.
[121] J. E. Smith and R. Nair, "The architecture of virtual machines," *Computer*, vol. 38, no. 5, pp. 32–38, May 2005.
[122] M. Pearce, S. Zeadally, and R. Hunt, "Virtualization: Issues, security threats, and solutions," *ACM Comput. Surv.*, vol. 45, no. 2, pp. 1–39, Feb. 2013, doi: 10.1145/2431211.2431216.
[123] V. Mauch, M. Kunze, and M. Hillenbrand, "High performance cloud computing," *Future Gener. Comput. Syst.*, vol. 29, no. 6, pp. 1408–1416, 2013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X12000647
[124] (2009). *Security Guidance for Critical Areas of Focus in Cloud Computing V2.1*. [Online]. Available: https://cloudsecurityalliance.org/csaguide.pdf
[125] I. Sadooghi *et al.*, "Understanding the performance and potential of cloud computing for scientific applications," *IEEE Trans. Cloud Comput.*, vol. 5, no. 2, pp. 358–371, Apr. 2017.
[126] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," in *Proc. 6th Conf. Symp. Operating Syst. Design Implement.*, vol. 6. New York, NY, USA, 2004, p. 10.
[127] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008, doi: 10.1145/1327452.1327492.
[128] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The Google file system," *ACM SIGOPS Operating Syst. Rev.*, vol. 37, no. 5, pp. 29–43, Dec. 2003, doi: 10.1145/1165389.945450.
[129] J. Ekanayake, S. Pallickara, and G. Fox, "MapReduce for data intensive scientific analyses," in *Proc. IEEE 4th Int. Conf. eScience*, Dec. 2008, pp. 277–284.
[130] M. Zaharia, *An Architecture for Fast and General Data Processing on Large Clusters* (Association for Computing Machinery). San Rafael, CA, USA: Morgan & Claypool, 2016.
[131] T. Gunarathne, T.-L. Wu, J. Qiu, and G. Fox, "MapReduce in the clouds for science," in *Proc. IEEE 2nd Int. Conf. Cloud Comput. Technol. Sci.*, Nov. 2010, pp. 565–572.
[132] S. Ostermann, A. Iosup, N. Yigitbasi, R. Prodan, T. Fahringer, and D. Epema, "A performance analysis of EC2 cloud computing services for scientific computing," in *Cloud Computing*, D. R. Avresky, M. Diaz, A. Bode, B. Ciciani, and E. Dekel, Eds. Berlin, Germany: Springer, 2010, pp. 115–131.
[133] J. Dongarra and P. Luszczek, *HPC Challenge Benchmark*. Boston, MA, USA: Springer, 2011, pp. 844–850, doi: 10.1007/978-0-387-09766-4_156.
[134] A. Iosup, S. Ostermann, M. N. Yigitbasi, R. Prodan, T. Fahringer, and D. H. J. Epema, "Performance analysis of cloud computing services for many-tasks scientific computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 22, no. 6, pp. 931–945, Jun. 2011.
[135] P. Mehrotra *et al.*, "Performance evaluation of Amazon EC2 for NASA HPC applications," in *Proc. 3rd Workshop Sci. Cloud Comput.*, New York, NY, USA, 2012, pp. 41–50, doi: 10.1145/2287036.2287045.
[136] D. H. Bailey *et al.*, "The NAS parallel benchmarks," *Int. J. Supercomput. Appl.*, vol. 5, no. 3, pp. 63–73, Sep. 1991, doi: 10.1177/109434209100500306.
[137] A. Petitet, R. Whaley, J. Dongarra, and A. Cleary. (Dec. 2018). *HPL—A Portable Implementation of the High-Performance Linpack Benchmark for Distributed-Memory Computers*. [Online]. Available: http://www.netlib.org/benchmark/hpl/
[138] G. Wang and T. S. E. Ng, "The impact of virtualization on network performance of Amazon EC2 data center," in *Proc. IEEE INFOCOM*, Mar. 2010, pp. 1–9.
[139] J. R. Lange *et al.*, "Minimal-overhead virtualization of a large scale supercomputer," in *Proc. 7th ACM SIGPLAN/SIGOPS Int. Conf. Virtual Execution Environments*, New York, NY, USA, 2011, pp. 169–180. [Online]. Available: https://doi.org/10.1145/1952682.1952705
[140] P. Leitner and J. Cito, "Patterns in the chaos—A study of performance variation and predictability in public iaas clouds," *ACM Trans. Internet Technol.*, vol. 16, no. 3, p. 15, Apr. 2016, doi: 10.1145/2885497.
[141] A. Gupta *et al.*, "Evaluating and improving the performance and scheduling of HPC applications in cloud," *IEEE Trans. Cloud Comput.*, vol. 4, no. 3, pp. 307–321, Jul. 2016.
[142] S. Sehrish, G. Mackey, P. Shang, J. Wang, and J. Bent, "Supporting HPC analytics applications with access patterns using data restructuring and data-centric scheduling techniques in MapReduce," *IEEE Trans. Parallel Distrib. Syst.*, vol. 24, no. 1, pp. 158–169, Jan. 2013.
[143] L. S. Blackford *et al.*, "An updated set of basic linear algebra subprograms (BLAS)," *ACM Trans. Math. Softw.*, vol. 28, no. 2, pp. 135–151, 2002.
[144] M. P. Forum, "MPI: A message-passing interface standard," Univ. Tennessee, Knoxville, TN, USA, Tech. Rep. CS-94-230, 1994.
[145] A. Krizhevsky, "One weird trick for parallelizing convolutional neural networks," 2014, *arXiv:1404.5997*. [Online]. Available: https://arxiv.org/abs/1404.5997
[146] T. Ben-Nun and T. Hoefler, "Demystifying parallel and distributed deep learning: An in-depth concurrency analysis," 2018, *arXiv:1802.09941*. [Online]. Available: https://arxiv.org/abs/1802.09941
[147] E. Frank, M. A. Hall, G. Holmes, R. Kirkby, B. Pfahringer, and I. H. Witten, *Weka—A Machine Learning Workbench for Data Mining*. Berlin, Germany: Springer, 2005, pp. 1305–1314. [Online]. Available: http://researchcommons.waikato.ac.nz/handle/10289/1497
[148] T. Guo, "Cloud-based or on-device: An empirical study of mobile deep inference," in *Proc. IEEE Int. Conf. Cloud Eng. (IC2E)*, Apr. 2018, pp. 184–190.
[149] A.-K. Koliopoulos, P. Yiapanis, F. Tekiner, G. Nenadic, and J. Keane, "A parallel distributed Weka framework for big data mining using spark," in *Proc. IEEE Int. Congr. Big Data*, Jun. 2015, pp. 9–16.
[150] M. Assefi, E. Behravesh, G. Liu, and A. P. Tafti, "Big data machine learning using Apache spark MLlib," in *Proc. IEEE Int. Conf. Big Data (Big Data)*, Dec. 2017, pp. 3492–3498.
[151] Y. You, I. Gitman, and B. Ginsburg, "Large batch training of convolutional networks," 2017, *arXiv:1708.03888*. [Online]. Available: http://arxiv.org/abs/1708.03888
[152] S. L. Smith, P.-J. Kindermans, C. Ying, and Q. V. Le, "Don't decay the learning rate, increase the batch size," in *Proc. Int. Conf. Learn. Represent.*, 2018, pp. 1–11.
[153] J. Dowling, "Distributed deep learning with Apache spark and tensorflow," in *Proc. Eur. Spark+AI Summit*, 2018, pp. 1–48.
[154] A. Paszke *et al.*, "Pytorch: An imperative style, high-performance deep learning library," in *Adv. Neural Inf. Process. Syst.*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d' Alché-Buc, E. Fox, and R. Garnett, Eds. Red Hook, NY, USA: Curran Associates, 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf
[155] F. Niu, B. Recht, C. Ré, and S. Wright, "Hogwild!: A lock-free approach to parallelizing stochastic gradient descent," in *Proc. NIPS*, vol. 24, Jun. 2011, pp. 1–22.
[156] J. J. Dai *et al.*, "BigDL: A distributed deep learning framework for big data," in *Proc. ACM Symp. Cloud Comput.*, 2019, pp. 50–60. [Online]. Available: https://arxiv.org/pdf/1804.05839.pdf
[157] M. Li *et al.*, "Scaling distributed machine learning with the parameter server," in *Proc. 11th USENIX Symp. Operating Syst. Design Implement.*, New York, NY, USA, 2014, pp. 583–598.
[158] N. Nawi, M. Ransing, and R. Ransing, "An improved learning algorithm based on the Broyden-Fletcher-Goldfarb-Shanno (BFGS) method for back propagation neural networks," in *Proc. 6th Int. Conf. Intell. Syst. Design Appl.*, vol. 1, Oct. 2006, pp. 152–157.
[159] S. Moreno-Álvarez, J. M. Haut, M. E. Paoletti, J. A. Rico-Gallego, J. C. Díaz-Martín, and J. Plaza, "Training deep neural networks: A static load balancing approach," *J. Supercomput.*, vol. 76, no. 12, pp. 9739–9754, Dec. 2020, doi: 10.1007/s11227-020-03200-6.

ABOUT THE AUTHORS



Juan M. Haut (Senior Member, IEEE) received the B.Sc. and M.Sc. degrees in computer engineering and the Ph.D. degree in information technology from the University of Extremadura, Cáceres, Spain, in 2011, 2014, and 2019, respectively, supported by the University Teacher Training Programme from the Spanish Ministry of Education.

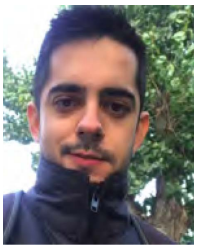
He was a member of the Hyperspectral Computing Laboratory (HyperComp), Department of Technology of Computers and Communications, University of Extremadura. He is also an Associate Professor with the Department of Communication and Control Systems, National Distance Education University, Madrid, Spain. His research interests include remote sensing data processing and high-dimensional data analysis, applying machine (deep) learning, and cloud computing approaches. In these areas, he has authored/coauthored more than 30 JCR journal articles (more than 20 in IEEE journals) and 20 peer-reviewed conference proceeding papers. Some of his contributions have been recognized as hot-topic publications for their impact on the scientific community.

Dr. Haut was a recipient of the Outstanding Ph.D. Award from the University of Extremadura in 2019. He was a recipient of the Outstanding Paper Award at the 2019 IEEE WHISPERS Conference. From his experience as a reviewer, it is worth mentioning his active collaboration in more than ten scientific journals, such as IEEE TRANSACTIONS ON GEOSCIENCE AND REMOTE SENSING, IEEE JOURNAL OF SELECTED TOPICS IN APPLIED EARTH OBSERVATIONS AND REMOTE SENSING, and IEEE GEOSCIENCE AND REMOTE SENSING LETTERS, being awarded with the Best Reviewer recognition of IEEE GEOSCIENCE AND REMOTE SENSING LETTERS in 2018. Furthermore, he has guest-edited three special issues on hyperspectral remote sensing for different journals. He is also an Associate Editor of IEEE GEOSCIENCE AND REMOTE SENSING LETTERS and IEEE JOURNAL ON MINIATURIZATION FOR AIR AND SPACE SYSTEMS.

Mercedes E. Paoletti (Senior Member, IEEE) received the B.Sc. and M.Sc. degrees in computer engineering from the University of Extremadura, Cáceres, Spain, in 2014 and 2016, respectively, and the Ph.D. degree, supported by the University Teacher Training Programme from the Spanish Ministry of Education, from the University of Extremadura in 2020.

She was a member of the Hyperspectral Computing Laboratory (HyperComp), Department of Technology of Computers and Communications, University of Extremadura. She is currently a Researcher with the Department of Computer Architecture, University of Málaga, Málaga, Spain. Her research interests include remote sensing and analysis of very high spectral resolution with the current focus on deep learning (DL) and high-performance computing.

Dr. Paoletti was a recipient of the 2019 Outstanding Paper Award Recognition at the IEEE WHISPERS 2019 Conference and the Outstanding Ph.D. Award at the University of Extremadura in 2020. She has served as a Reviewer for IEEE TRANSACTIONS ON GEOSCIENCE AND REMOTE SENSING and IEEE GEOSCIENCE AND REMOTE SENSING LETTERS, in which she was recognized as a best reviewer in 2019.



Sergio Moreno-Álvarez received the B.Sc. and M.Sc. degrees in computer engineering from the University of Extremadura, Cáceres, Spain, in 2017 and 2019, respectively, where he is currently working toward the Ph.D. degree at the Department of Computer Systems Engineering and Telematics.

As research experience, he has participated in regional projects. He is currently a Researcher with the School of Technology, University of Extremadura. He has published four JCR articles in international journals and three presentations at international and national conferences. His main interests are high-performance computing, neural networks, and deep learning (DL).

Javier Plaza (Senior Member, IEEE) received the M.Sc. and Ph.D. degrees in computer engineering from the University of Extremadura, Cáceres, Spain, in 2004 and 2008, respectively.

He is currently a member of the Hyperspectral Computing Laboratory, Department of Technology of Computers and Communications, University of Extremadura. He has authored more than 200 publications, including over 80 JCR journal articles, ten book chapters, and 100 peer-reviewed conference proceeding papers. His main research interests comprise hyperspectral data processing and parallel computing of remote sensing data.

Dr. Plaza was a recipient of the Outstanding Ph.D. Dissertation Award at the University of Extremadura in 2008. He was also a recipient of the Best Column Award of *IEEE Signal Processing Magazine* in 2015 and the Most Highly Cited Paper (2005–2010) in the *Journal of Parallel and Distributed Computing*. He received the best paper awards at the IEEE International Conference on Space Technology and the IEEE Symposium on Signal Processing and Information Technology. He has guest-edited four special issues on hyperspectral remote sensing for different journals. He is also an Associate Editor for IEEE GEOSCIENCE AND REMOTE SENSING LETTERS and IEEE Remote Sensing Code Library. Additional information: http://www.umbc.edu/rssipl/people/jplaza.

Juan-Antonio Rico-Gallego received the Computer Science Engineering degree and the Ph.D. degree in computer science from the University of Extremadura, Cáceres, Spain, in 2002 and 2016, respectively.

He was a software consultant. He is currently an Associate Professor with the Department of Computer Systems Engineering, University of Extremadura. His research interests are in analytical performance models on heterogeneous platforms and the usage of deep and reinforcement learning techniques to high-performance computing (HPC) platform problems, including scheduling, process deployment and mapping, load balancing, and communication modeling. He codevelops AzequiaMPI, an efficient thread-based full MPI 1.3 standard implementation. His other research interests include current message passing interface (MPI) implementations and applications.

Antonio Plaza (Fellow, IEEE) received the M.Sc. and Ph.D. degrees in computer engineering from the University of Extremadura, Cáceres, Spain, in 1999 and 2002, respectively.

He is currently the Head of the Hyperspectral Computing Laboratory, Department of Technology of Computers and Communications, University of Extremadura. He has authored more than 600 publications, including over 300 JCR journal articles (over 220 in IEEE journals), 23 book chapters, and around 300 peer-reviewed conference proceeding papers. His main research interests comprise hyperspectral data processing and parallel computing of remote sensing data.

Prof. Plaza was a member of the Steering Committee of IEEE JOURNAL OF SELECTED TOPICS IN APPLIED EARTH OBSERVATIONS AND REMOTE SENSING (JSTARS). He is also a Fellow of IEEE "for contributions to hyperspectral data processing and parallel computing of Earth observation data." He was a recipient of the Best Column Award of *IEEE Signal Processing Magazine* in 2015, the 2013 Best Paper Award of the JSTARS, and the Most Highly Cited Paper (2005–2010) in the *Journal of Parallel and Distributed Computing*. He received best paper awards at the IEEE International Conference on Space Technology and the IEEE Symposium on Signal Processing and Information Technology. He was a recipient of the recognition of Best Reviewers of IEEE GEOSCIENCE AND REMOTE SENSING LETTERS in 2009 and the recognition of Best Reviewers of IEEE TRANSACTIONS ON GEOSCIENCE AND REMOTE SENSING in 2010, for which he served as Associate Editor from 2007 to 2012. He has guest-edited ten special issues on hyperspectral remote sensing for different journals. He is also an Associate Editor for IEEE ACCESS (receiving recognition as an Outstanding Associate Editor of the journal in 2017). He has served as the Director of Education Activities for the IEEE Geoscience and Remote Sensing Society (GRSS) from 2011 to 2012 and the President of the Spanish Chapter of the IEEE GRSS from 2012 to 2016. He has reviewed more than 500 manuscripts for over 50 different journals. He has served as the Editor-in-Chief of IEEE TRANSACTIONS ON GEOSCIENCE AND REMOTE SENSING from 2013 to 2017. He is also the Editor-in-Chief of IEEE JOURNAL ON MINIATURIZATION FOR AIR AND SPACE SYSTEMS (J-MASS). He has been distinguished as a Highly Cited Researcher by Clarivate Analytics from 2018 to 2020. Additional information: http://www.umbc.edu/rssipl/people/aplaza.