electronics

MDPI

Article

Performance of Two Approaches of Embedded Recommender Systems

Francisco Pajuelo-Holguera [1], Juan A. Gómez-Pulido [1,*] and Fernando Ortega [2]

[1] Department of Tecnología de Computadores y Comunicaciones, Escuela Politécnica, Universidad de Extremadura, 10003 Cáceres, Spain; franciscoph@unex.es

[2] Department of Sistemas Informáticos, ETSI Sistemas Informáticos, Universidad Politécnica de Madrid, 28031 Madrid, Spain; fernando.ortega@upm.es

* Correspondence: jangomez@unex.es

Received: 22 February 2020; Accepted: 21 March 2020; Published: 25 March 2020

Abstract: Nowadays, highly portable and low-energy computing environments require programming applications able to satisfy computing time and energy constraints. Furthermore, collaborative filtering based recommender systems are intelligent systems that use large databases and perform extensive matrix arithmetic calculations. In this research, we present an optimized algorithm and a parallel hardware implementation as good approach for running embedded collaborative filtering applications. To this end, we have considered high-level synthesis programming for reconfigurable hardware technology. The design was tested under environments where usual parameters and real-world datasets were applied, and compared to usual microprocessors running similar implementations. The performance results obtained by the different implementations were analyzed in computing time and energy consumption terms. The main conclusion is that the optimized algorithm is competitive in embedded applications when considering large datasets and parallel implementations based on reconfigurable hardware.

Keywords: embedded systems; collaborative filtering; recommender systems; parallelism; reconfigurable hardware; high-level synthesis

1. Introduction

Nowadays, in the framework of the information society, a large amount of information is being generated from multiple and heterogeneous data sources. The own interaction of the user who generates or uses this information is added to the same. Representative examples can be found in areas such as e-commerce (users who buy and value products) and the entertainment industry (users who value series and movies). This information is usually stored in large databases, permanently and dynamically growing and updating, which constitute a source of knowledge regarding user behavior, so that predictions and recommendations can be made. This is where recommendation systems emerge.

Recommender Systems (RS) [1] are algorithmic techniques that allow users to obtain recommendations and predictions after an intelligent processing of the data of large databases. RS give personalized recommendations to the users according to their behavior when requesting and handling information [2,3]. In this sense, RS are also known as filters because they block the data not connected to the users' behavior.

Besides the analysis and recommendation of information, an important application of RS is the prediction of the users' behavior. For example, in the *Predicting Student Performance* (PSP) problem [4], the score of an evaluation task in the academic environment for a particular student can be predicted when RS considers it as a ranking prediction problem. Nevertheless, the most popular implementation

of RS is *Collaborative Filtering* (CF) [5,6], where users with similar preferences in the past will have similar preferences in the future [7]. For example, if two users have rated the same movies as positive, new movies that either rates as positive might be liked by the other user.

A matrix defines the relationship between users and items in CF. This matrix stores the ratings (explicit or implicit) of the users to the items, and has a high level of sparsity, because users only rate a small number of available items. Popular online applications, such as e-commerce websites or movies databases, generate rating matrices composed of thousands of million ratings, where hundreds of thousands of users have rated hundreds of thousands of items.

The way to fill the gaps of the sparse ratings matrix [8] considers the *Matrix Factorization* (MF) technique [9]. MF generates a scalable model for prediction purposes [10] composed of two matrices. The prediction is a combination of factors as result of multiplying the row corresponding to a user in the user-latent space with the column corresponding to an item in the item-latent matrix. In addition, MF assumes that users' ratings are conditioned by K latent factors describing the items of the system. MF algorithms try to find these hidden factors through the rating matrix.

We would like to highlight the interest in implementing a CF algorithm in hardware for running embedded applications due to several reasons. Firstly, we must bear in mind that CF involves large amount of data because of the number of users and items in databases. The needs of predictions and data handling involve high computational efforts, especially if real time constraints are required. Therefore, the design of hardware circuits that accelerate some processes of the algorithm is especially interesting. Besides, possible embedded applications of CF require fast algorithms if they should be performed on small, low-power computing environments. Therefore, we focus the research on implementing embedded applications of CF by considering *Field Programmable Gate Array* (FPGA) devices [11], under the *Reconfigurable Computing* (RC) [12] and System-On-Chip (SoC) [13] concepts.

We propose using FPGA devices for designing accelerated CF algorithms because this technology combines software flexibility with hardware performance by exploiting parallelism. Thus, if an embedded implementation is designed carefully by following these advantages, it can provide excellent results, even surpassing the performance delivered by usual microprocessors or *Central Processing Units* (CPU) in similar experimental conditions [14]. Other design approaches based on different hardware technologies can also be explored. In this sense, *Graphical Processing Units* (GPUs) can be programmed by using OpenCL for similar purposes, although their high power consumption could be a constraint when using them for embedded applications.

In summary, our proposal is to design an embedded, low-energy implementation of an efficient CF algorithm in order to perform applications on highly-portable light computing environments. Our approach was successfully tested considering several state-of-the-art datasets.

The remainder of this paper is structured as follows. We present some related works in Section 2. In Section 3, we discuss the basis of two approaches, basic and enhanced, of CF algorithms. Next, Section 4 explains the design and implementation of both algorithms, emphasizing on the parallelization strategy considered for improving the performance results. Section 5 shows a performance comparison between the two approaches and usual microprocessors, detailing the state-of-the-art datasets considered, the experimental procedure followed, and the timing and power results. Finally, the conclusions of this paper are summarized in Section 6.

2. Related Works

RS are a good opportunity to provide advanced services to Internet users. Some classic examples of heterogeneous successful applications are PHOAKS [15] (it helps users to locate useful information on the *World Wide Web* (WWW) examining USenet news messages), Referral Web [16] (it combines social networks and collaborative filtering), Fab [17] (it combines content-based information with collaborative filtering), Siteseer [18] (a conceptual recommender system for CiteSeerX), and many others. However, currently growing concepts in the Internet domain, such as Internet of Things, autonomous driving, and augmented reality, among many others, are pushing to consider

new applications of the RS. For example, we can find novel and advanced applications of RS in vehicles [19], voice-enabled devices [20], smartphones [21], and multimedia data for robustness [22], diversification [23], and real-time [24] recommendation aims, among many other examples.

In the context of an increasing application of the RS, many research efforts are focused on improving the accuracy and reducing their limitations. In this regard, RS have some limitations, especially related to their complexity and difficulty in understanding them. They represent black boxes that require personalized explanations related to the individuals' mental models [25], which has consequences in many areas, such as computer vision [26].

Computing systems based on low-performance and low-consumption microprocessors may be involved in some of these new fields of application of RS. Thus, there are environments where RS could run on such computer systems, for example smartphones and IoT devices. In fact, the demand of computing resources by RS may have limited their application in these areas and devices. Particularly, mobile RS are an interesting area for online applications (social networks, e-commerce, and streaming platforms) in situations where the data volume can produce overload. These situations may occur more and more frequently, given the rapid increase in the use of mobile devices in a context of continuous growth and improvement of network infrastructure. The links between web and mobile RS are identified in [27] to provide guidelines for embedded RS in mobile domain. We find some examples of mobile RS in recommending different types of media to its users using a context-aware approach [28] or in recommending photos by means of current contextual data in combination with information found in the photos [29]. Other examples of mobile RS can be found in the mobile news based on the current context and format [30], the recommendation of music depending on the daily activities of a person [31], or the passengers of a car [32].

For all the above reasons, the tools and technologies for designing and implementing embedded computing systems based on low-consumption devices can lead to the application of RS for many purposes in novel fields. Our proposal considers the reconfigurable technology based on FPGA devices for implementing fast, low-power collaborative filtering algorithms for embedded applications. This proposal is in line with other works where ML functions and features have been implemented using similar technology, for different purposes, mainly for acceleration tasks. Thus, we can find FPGA technology applied for *Convolutional Neural Networks* (CNN) [33], *Deep Learning* (DL) [34], K-Means clustering [35–37], and kernel density estimation [38], among others.

It is particularly interesting to explore the application of FPGAs for CF, especially for acceleration purposes. In this regard, there are some attempts to accelerate tasks involved in cloud services and large databases, such as Amazon [39]. We can find some examples of FPGA implementations of different aspects of RS algorithms, rather than the whole system itself. For example, a *Stochastic Gradient Descent* (SGD) algorithm [40] used for training some RS models is implemented on FPGA considering single-precision floating-point [41]. In this sense, our proposal takes a step forward, as we undertake the complete implementation of two CF algorithms, which are capable of handling real datasets.

3. Recommender Systems: Two Approaches

In this section, we present two approaches of CF algorithms, detailing their mathematical descriptions and how they work.

3.1. Basic Algorithm

In the context of machine learning, MF technique represents a well known family of algorithms that split a matrix $X \in \mathbb{R}^{n \times m}$ into two matrices $U \in \mathbb{R}^{n \times k}$ and $V \in \mathbb{R}^{k \times m}$, in such a way that $X \approx U \cdot V$ [42]. Note that the rank of the matrices U and V is much smaller than the rank of X, since $k \ll n$ and $k \ll m$. Therefore, the factorized matrices U and V contain a compact representation of the original matrix X.

Applied to CF, MF based RS factorize the sparse rating matrix $R \in \mathbb{R}^{n \times m}$ that contains the set of known ratings of n users to m items [43]. The fundamental assumption of these kinds of algorithms is that the ratings of the users to the items are conditioned by a subset of latent factors intrinsic to the users and items. For example, in a movies' RS, it is assumed that the rating a user provides to a movie is conditioned by the genre of that movie. As consequence of the factorization process, two new matrices are generated: $P \in \mathbb{R}^{n \times k}$, which represents the k-latent factors of the n users; and $Q \in \mathbb{R}^{m \times k}$, which represents the k-latent factors of the m items. Once the factorization is performed, the rating predictions ($\hat{r}_{ui}$) of a user u to an item i can be computed by the dot product of the row vector of the matrix P that contains the latent factors of the user u ($\vec{p}_u$) and the column vector of the matrix Q that contains the latent factors of the item i ($\vec{q}_i$):

$$\hat{r}_{ui} = \vec{p}_u \cdot \vec{q}_i^T. \tag{1}$$

Hence, the learning process consists on find the optimal parameters for the matrices P and Q that verifies

$$R \approx P \cdot Q^T. \tag{2}$$

This process is usually raised as an optimization problem in which the quadratic difference between the known ratings ($r_{u,i}$) of the matrix R and the predicted ones ($\vec{p}_u \cdot \vec{q}_i^T$) must be minimized:

$$\min_{\vec{p}_u, \vec{q}_i} \sum_{(u,i) \in R} (r_{u,i} - \vec{p}_u \cdot \vec{q}_i^T)^2. \tag{3}$$

The most popular implementation of MF applied to CF is Probabilistic Matrix Factorization (PMF) [44]. PMF performs the factorization thorough a probabilistic model that represents interaction between the users and items in a CF context. Figure 1 contains a graphical representation of this probabilistic model. The figure contains three representational elements: circles that symbolize random variables; arrows between two variables that indicate dependence between that random variables; and rectangles that indicate repetitions of the random variables. The color of the circles indicates if the random variables are observed (black) or must be learned (white). As we can observe, there exists three random variable: R_{ui} that symbolizes the rating of the user u to the item i; P_u that symbolizes the latent factors of each user u; and Q_i that symbolizes the latent factors of each item i. The arrows between P_u and Q_i with R_{ui} denote that there exists dependency between the rating of user u to item i and the latent factors of user u and item i. PMF assumes a Gaussian distribution for all the random variables. σ_R, σ_P and σ_Q denotes model hyper-parameters.

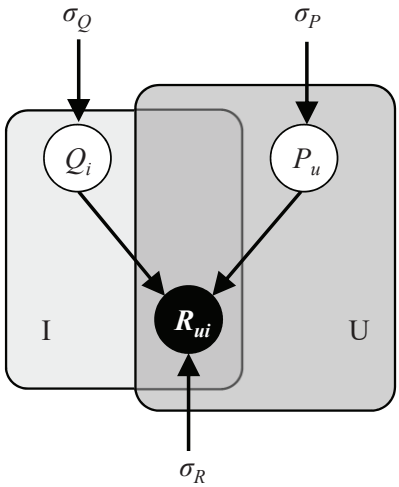

Figure 1. Graphical representation of PMF model.

Algorithm 1 summarizes PMF. The inputs are the rating matrix R, the number of latent factors K, and the hyper-parameters to control the learning process λ and γ. The outputs are the latent factors matrices P and Q learned from the rating matrix.

Algorithm 1: PMF algorithm.

input : R, K, λ, γ
output: P, Q
Create a random matrix P with U rows and K columns
Create a random matrix Q with I rows and K columns
repeat
  for *each user u* **do** // This loop can me parallelized for each user
    for *each item i rated by user u* **do**
      error = $R[u][i]$ - dotProduct($P[u]$, $Q[i]$)
      for *each factor k* **do**
        $P[u][k] += \gamma \cdot (error \cdot P[u][k] - \lambda \cdot Q[i][k])$
  for *each item i* **do** // This loop can be parallelized for each item
    for *each user u that has rated the item i* **do**
      error = $R[u][i]$ - dotProduct($P[u]$, $Q[i]$)
      for *each factor k* **do**
        $Q[i][k] += \gamma \cdot (error \cdot Q[i][k] - \lambda \cdot P[u][k])$
until *convergence*
return P, Q

3.2. BNMF Algorithm

Bayesian Non-negative Matrix Factorization (BNMF) [9] model is another factorization model designed for CF based RS. BNMF model has demonstrated its superiority by providing more accurate predictions and recommendations than PMF model. As PMF, BNMF factorizes the rating matrix in a probabilistic way.

The main objective of BNMF is to provide an understandable probabilistic meaning of the latent factors space generated as consequence of the factorization process. To achieve this, the model has been designed in such a way that it better represents the interaction between users and items. Instead of assuming a continuous distribution to represent ratings, such as Gaussian distribution, a discrete distribution is used. This coincides with the reality of most CF systems, where users must rate items on a pre-set scale (e.g., 1–5 stars).

Figure 2 contains a graphical representation of BNMF model. The model is composed by the following random variables:

- $\vec{\theta}_u$ is a K dimensional vector from a Dirichlet distribution. This random variables are used to represent the probability that a user belongs to each group.
- κ_{ik} from the Beta distribution used to represent the probability that a user in the group k likes the item i.
- Z_{ui} from the Categorical distribution used to represents that the user u rates the item i as if he or she belongs to the group k.
- ρ_{ui} from the Binomial distribution used to represent the observable rating of the user u to the item i.

The model also contains the following hyper-parameters:

- α is related to the possibility of obtaining overlapping groups of users sharing the same preferences.
- β is related to the amount of evidences required to belong to a group.

- K is related to the number of groups (i.e., number of latent factors) that exists in the dataset.
- R is related to the Binomial distribution which take values from 0 to R.

To be able to compute predictions with the BNMF model, we must determine the conditional probability distribution of the non-observable random variables given a set of observations (i.e., the known ratings). Applying the variational inference technique [45], we can obtain the algorithm to perform this task. Algorithm 2 contains a detailed explanation about the training phase of BNMF model. For further information about the inference process, see [9].

Algorithm 2: BNMF algorithm. The algorithm returns the latent factors for each user and item. Input ratings (r_{ui}) must be normalized.

input : r_{ui}, α, β, K, R
output: p_{uk}, q_{ik}
temp : γ_{uk}, ϵ^{-}_{ik}, ϵ^{+}_{ik}, λ_{uik}, λ'_{uik}
Initialize γ_{uk}
Initialize ϵ^{-}_{ik}
Initialize ϵ^{+}_{ik}
repeat
  for *each user u* **do**
    for *each item i rated by user u* **do**
      for *each factor k* **do**
        $\lambda'_{uik} \leftarrow exp(\Psi(\gamma_{uk}) + r^{+}_{ui} \cdot \Psi(\epsilon^{+}_{ik}) + r^{-}_{ui} \cdot \Psi(\epsilon^{-}_{ik}) - R \cdot \Psi(\epsilon^{+}_{ik} + \epsilon^{-}_{ik}))$
      for *each factor k* **do**
        $\lambda_{uik} \leftarrow \frac{\lambda'_{uik}}{\lambda'_{ui1} + \cdots + \lambda'_{uiK}}$
  for *each item i* **do**
    $\epsilon^{+}_{ik} \leftarrow \beta$
    $\epsilon^{-}_{ik} \leftarrow \beta$
  for *each user u* **do**
    $\gamma_{uk} \leftarrow \alpha$
    for *each item i rated by user u* **do**
      for *each factor k* **do**
        $\gamma_{uk} \leftarrow \gamma_{uk} + \lambda_{uik}$
        $\epsilon^{+}_{ik} \leftarrow \epsilon^{+}_{ik} + \lambda_{uik} \cdot R \cdot r_{ui}$
        $\epsilon^{-}_{ik} \leftarrow \epsilon^{-}_{ik} + \lambda_{uik} \cdot R \cdot (1 - r_{ui})$
until *convergence*
for *each factor k* **do**
  for *each user u* **do**
    $p_{uk} \leftarrow \frac{\gamma_{uk}}{\sum_{f=1..K} \gamma_{uf}}$
  for *each item i* **do**
    $q_{ik} \leftarrow \frac{\epsilon^{+}_{ik}}{\epsilon^{+}_{ik} + \epsilon^{-}_{ik}}$

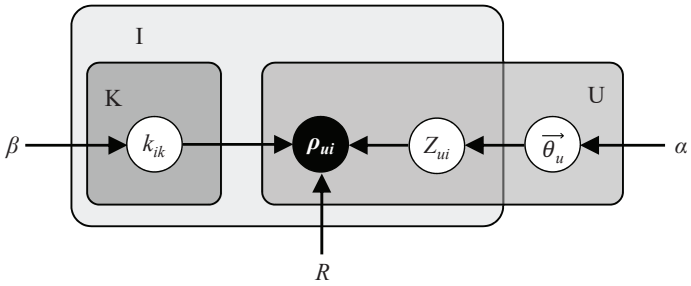

Figure 2. Graphical representation of BNMF model.

4. Hardware Designs for Embedded Applications

In this section, we present the hardware implementations of PMF and BNMF. The purpose of both implementations is twofold. On the one hand, the operations of the algorithms are accelerated by using the parallelism that hardware provides; on the other hand, the energy consumption is reduced in comparison with usual microprocessors.

4.1. PMF Design

PMF was parallelized by considering *High-Level Synthesis* (HLS) technology [46]. HLS transforms C specifications (C, C++, SystemC, or OpenCL code) into a *Register Transfer Level* (RTL) implementation, which allows us to synthesize the design to any Xilinx FPGA. This way, HLS facilitates the fast design of efficient circuits by parallelizing code automatically. Specifically, for this work, we considered Vivado HLS tool [47], which was deeply analyzed by O'Loughlin *et al.* [48].

The main parallelization strategy for PMF is described in [49]. As we can see in Algorithm 1, two consecutive loops can be parallelized after initialization in order to update the corresponding factorized matrices for each user/item. These loops are sequentially performed several times.

4.2. BNMF Design

In this section, we show how we implemented the BNMF algorithm on a reconfigurable hardware platform. Previously, we implemented two versions of PMF on FPGA. The first version was a simple design without parallelism in order to check the viability of using an embedded operating system for running a full recommender system and analyze its performance. The second version was a parallelized design in order to accelerate the operations. Therefore, next, we focused our efforts just on implementing a parallelized design of BNMF, once checked the viability of using the same hardware platforms and software tools applied to PMF. In this section, we detail how we designed the BNMF algorithm for a high-performance implementation on FPGA.

The main tools used for the design of the BNMF algorithm in an FPGA are summarized in Table 1. Zedboard is a low-energy and low-cost prototyping board that mounts a programmable System-on-Chip (SoC) including an ARM processing architecture. Furthermore, there are many elements and features to design any computing system based on Linux, Windows, and Android operating systems, among others, and interact with the user's needs.

Table 1. Main tools for implementing BNMF in FPGA.

Hardware	**Zedboard Zynq-7000**	***SoC*:** ***Elements*:** ***Memory*:** ***Oscillators*:**	**Xilinx Zynq XC7Z020** **HDMI, VGA, audio, Ethernet, SD, USB ...** **512 MB DDR3** **100 MHz and 33.3 MHz**
Operating System	Linaro OS		
Software	Xilinx Vivado HLS		

Figure 3 shows the architecture where BNMF is implemented and executed. This architecture basically consists of three elements, mutually communicated along an AXI bus: external memory, multiprocessor system, and programmable logic.

- The memory of type DDR3 can store up to 512 MB. It hosts the datasets that provide the users and items to the RS, as well as the main program to control the BNMF flow. Storing the datasets in the external memory instead of in the internal memory blocks of the FPGA frees up space in the programmable logic to implement the BNMF core. In addition, an AXI interface was chosen to implement parallel access to memory so as not to limit bandwidth excessively. However, very large datasets may exceed the available memory capacity; in this case, the dataset is hosted on the SD card together with Linaro OS.
- The multiprocessor system is based on an ARM Dual Cortex-A9. It just runs the main control program: basic operations for initializing and starting the BNMF core implemented in the FPGA, as well as getting and displaying the results returned by it.
- The SoC block implements the BNMF core. The main advantage of this block is the high parallelization level of the operations described in Algorithm 2. Thus, the expected performance of this design would be higher than the performance given by a simple sequential code in the same main control program.

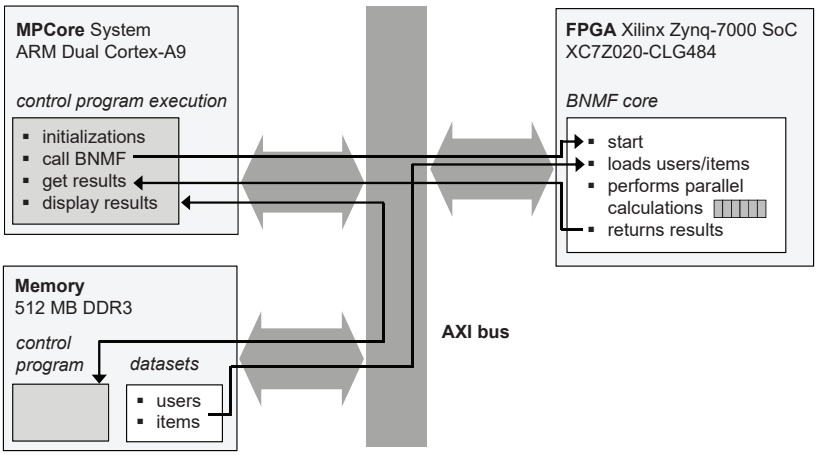

Figure 3. Basic architecture for BNMF on the Zynq Zedboard 7000.

As we did in PMF, we installed an embedded Linux OS (Linaro distribution) on the board in order to allow running the BNMF on the FPGA. This OS is launched from a separated partition in the SD card, thus the changes made by the program are written in that partition. The Linaro filesystem is a complete Ubuntu-based Linux distribution with graphical desktop. The advantage of using the Linaro is that we can work with the ZedBoard just as if we used a commercial processor. Thus, the code executed both in ZedBoard and in CPU is exactly the same.

4.3. Parallelization Strategy

In this section, we detail how the parallel implementation of the BNMF algorithm was designed. The results obtained in PMF encouraged us to improve the performance by designing a more accurate parallel design in BNMF.

The parallel design was developed mainly by programming with HLS. However, we also modified the design manually by including different optimization directives provided by HLS in order to increase the fine-grained parallelism without the need to modify the C code, in order to obtain a higher performance circuit. Thanks to these directives, we managed the way of parallelizing certain loops and operations. The most used directives were those for unrolling loops or functions, which allow

us to work with arrays in parallel. Additionally, other directives to later transfer data to the BNMF algorithm were used too.

Figure 4 allows explaining easily the parallelization strategy followed by the design. First, according to Algorithm 2, we perform random initializations of γ, e^+ and e^- in parallel, since they are matrices and are highly parallelizable.

Next, four consecutive blocks implement parallel operation for calculating some sections of the algorithm. These four blocks are executed sequentially because there is a clear data dependency between them.

The update of λ requires a great computational cost, since we could define it as a matrix vector. Basically, the parallelization consists in updating each of the elements of that matrix vector in parallel. Then, we also perform the update of e^+ and e^- in parallel. Finally, we calculate the user factors a and b in parallel.

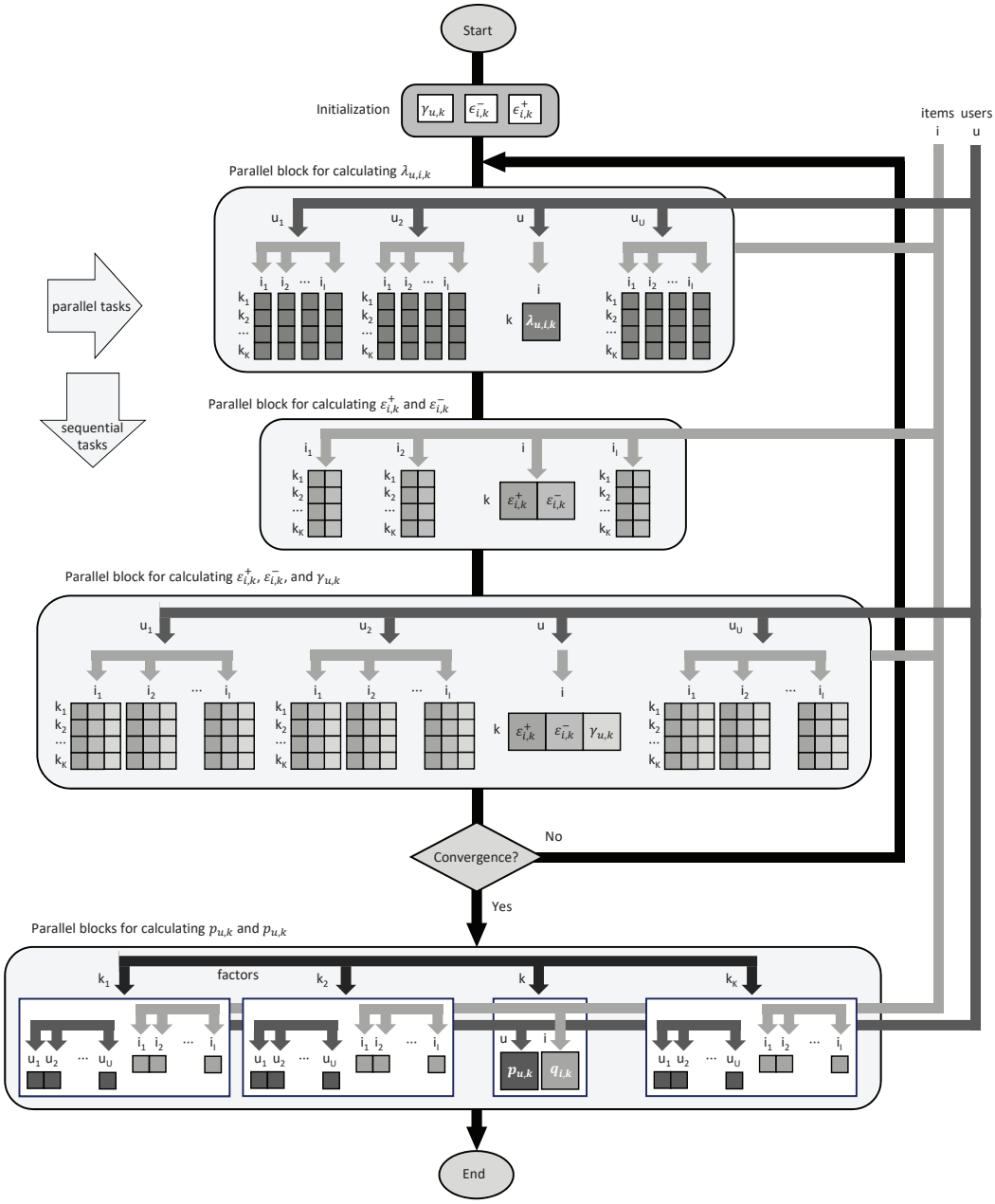

Figure 4. Strategy for parallelizing BNMF.

5. Performance Comparison

In this section, we highlight the different results obtained by PMF and BNMF. First, we explain the datasets considered for the experiments. Next, we show the performance results in terms of computing time and energy consumption.

5.1. Datasets

Both PMF and BNMF were tested using four state-of-the-art datasets of different characteristics, widely used for this purpose: The Movies Dataset (Kaggle), Movielens-100K, Movielens-1M, and Netflix-100M (Table 2). These datasets gather the activity of many users when rating movies with scores from 1 to 5, where each user rates at least 20 movies.

We chose datasets of very different sizes to check the impact of the matrix calculations in the performance given by the FPGA implementation. To get a rough idea, the product Users $\times$ Items goes from 6.3M in Kaggle to 8495M in Netflix-100M.

Table 2. Datasets used to test PMF and BNMF algorithms.

Dataset	Kaggle	Movielens-100k	Movielens-1M	Netflix-100M
Ratings	100,000	100,000	1,000,000	10,000,000
Users	700	943	6,000	480,188
Items	9000	1682	4000	17,691

5.2. Experimental Procedure

Figure 5 shows the phases of the experimental procedure followed in our research. First, we studied in depth the best way to parallelize BNMF, looking for those operations that can be parallelized without altering the right calculation of the remaining ones. Once the parallelizaton strategy was determined, we generated the parallel core using Xilinx Vivado HLS. The BNMF design was exported as IP core, which can be reached by the processor and memory in the architecture described in Figure 3. Next, this core was exported as bitstream into the Linaro OS, and the aforementioned datasets were added to perform the tests. Finally, the BNMF algorithm was executed and the results are validated.

This experimental procedure was performed as many times as different datasets available for performance purposes.

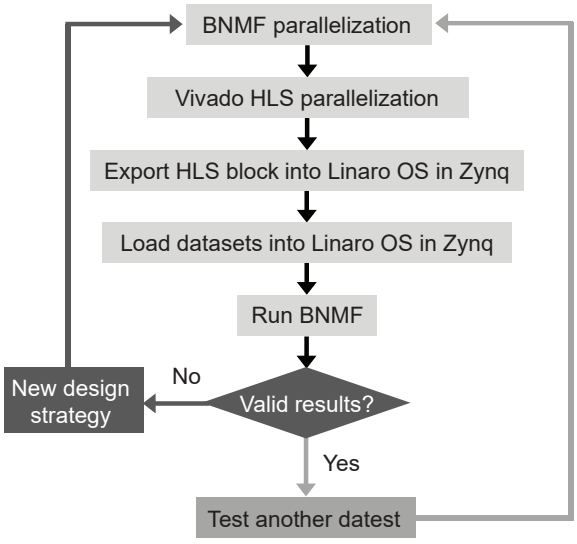

Figure 5. Experimental procedure.

5.3. Timing Results

In this section we show the computing time obtained by the hardware implementations of BNMF and PMF algorithms, and by an up-to-date microprocessor for comparison purposes.

With regard to the FPGA implementation, we measured the elapsed time by using HLS, considering the same FPGA device and the required operational frequency. Once the design is synthesized, HLS allows us to know whether the given frequency can be supported by the FPGA device, as well as the number of clock cycles used by the hardware. Hence, we calculated the elapsed computing time.

We considered for the CPU experiments an Intel i7-950 with clock frequency of 3 GHz. Note that the RS implemented on the FPGA reached a very low frequency compared to the CPU: 667 MHz. The CPU runs codes that implement the same operations described in the PMF and BNMG algorithms, considering the same parameters and datasets.

Table 3 shows the computing time in seconds of the PMF and BNMF algorithms for the CPU and FPGA implementations, considering the four datasets. We deduce two interesting conclusions.

Table 3. Computing time (s) and FPGA speedup of PMF and BNMF algorithms for the CPU and FPGA implementations.

Dataset	**Kaggle**		**Movielens-100k**		**Movielens-1M**		**Netflix-100M**	
Algorithm	**PMF**	**BNMF**	**PMF**	**BNMF**	**PMF**	**BNMF**	**PMF**	**BNMF**
CPU (s)	76.12	284.38	33.62	152.22	113.41	504.01	96,381.80	405,843.84
FPGA (s)	1129.70	313.82	831.04	163.72	2934.57	105.93	98,649.80	50,625.32
FPGA speedup	×0.07	×0.91	×0.04	×0.93	×0.04	×**4.76**	×0.98	×**8.02**

First, comparing both algorithms, we can observe that BNMF takes more computing time than PMF in CPU, although much less in FPGA. The reason is simply that BNMF provides the greatest parallelization degree in the FPGA implementation. Second, we can observe that, the larger is the dataset, the better are the results we obtain in the parallel implementation of BNMF in FPGA. In both the Kaggle dataset and the Movielenes-100k dataset, the time results are very similar. However, the two largest datasets begin to show a greater computing time difference between FPGA and CPU. Thus, for the Movielens-1M dataset, the FPGA gets a speedup of almost ×5, while this speedup increases to ×8 for the Netflix-100M dataset.

In conclusion, a FPGA implementation is more attractive for the BNMF algorithm and larger datasets. As a proposal, it would be interesting to experiment with larger sets corresponding to other types of data.

5.4. Power Results

Energy consumption is another important metric for computing systems performance. The RS algorithms have a certain energy impact on the hardware platforms. Knowing this impact is important because it helps us to optimize energy-aware designs of embedded RS. We keep in mind that embedded RS can be demanded for computing-intensive cases when performing many predictions over time.

Xilinx Vivado provides the total on-chip power of the FPGA implementations. Table 4 shows the power in watts of the PMF and BNMF algorithms for the CPU and FPGA implementations, considering the four datasets. We observe that the power reduction in any FPGA implementation is very high (more than 80% on average). Therefore, a clear advantage of implementing RS in FPGA is the low energy consumption with regard to current CPUs.

Under the algorithmic point of view, we can check in Table 4 that BNMF gives a more significant power reduction than PMF. This fact, along with the computing time reduction for large datasets deduced from Table 3, encourage us to consider BNMF as the best algorithmic option for building embedded RS applications.

Table 4. Power (w) and FPGA power reduction of PMF and BNMF algorithms for the CPU and FPGA implementations.

Dataset	**Kaggle**		**Movielens-100k**		**Movielens-1M**		**Netflix-100M**	
Algorithm	**PMF**	**BNMF**	**PMF**	**BNMF**	**PMF**	**BNMF**	**PMF**	**BNMF**
CPU (w)	8.21	11.33	7.33	10.81	12.31	16.26	32.21	41.20
FPGA (w)	0.95	2.52	0.82	2.24	1.64	4.41	3.03	7.37
FPGA power reduction	88%	78%	89%	80%	87%	73%	91%	83%

6. Conclusions

We researched the performance of two different approaches of collaborative filtering based recommender systems for embedded applications. For this purpose, we parallelized some operations by considering high-level synthesis technology for FPGA devices. Regarding computing time, the FPGA implementation of the Bayesian non-negative matrix factorization algorithm provided good speedups compared to general-purpose microprocessors when dealing with large datasets, and it surpassed clearly the results obtained by the probabilistic matrix factorization approach. Furthermore, the low power consumption of FPGA devices makes interesting the line of exploring computing solutions for embedded applications of collaborative filtering. In summary, the proposed approach allows running efficient embedded collaborative filtering applications when using low-energy computing systems based on FPGAs, taking advantage of the opportunity provided by reconfigurable computing to exploit parallelism.

Author Contributions: F.O. proposed the recommender systems algorithms and provided the datasets. F.P.-H. and J.A.G.-P. proposed the methodology and tools for designing the embedded architectures. F.P.-H. programmed the parallel codes, implemented the circuits, and measured the timing and energy behaviors. All authors analyzed the results, suggested the conclusions, and revised the manuscript. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Government of Extremadura (Spain) grant number IB16002 and by the AEI (State Research Agency, Spain) and the ERDF (European Regional Development Fund, EU) grant number TIN2016-76259-P. The APC was funded by the Government of Extremadura (Spain) grant number IB16002.

Conflicts of Interest: The authors declare no conflict of interest. The founding sponsors had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, and in the decision to publish the results.

Abbreviations

The following abbreviations are used in this manuscript:

BNMF	Bayesian Non-negative Matrix Factorization
CF	Collaborative Filtering
CNN	Convolutional Neural Networks
DL	Deep Learning
FPGA	Field-Programmable Gate Array
HLS	High-Level Synthesis
MF	Matrix Factorisation
ML	Machine Learning
PMF	Probabilistic Matrix Factorization
RC	Reconfigurable Computing
RMSE	Root Mean Squared Error
RS	Recommender Systems
RTL	Register Transfer Level
SGD	Stochastic Gradient Descent
SoC	System-on-Chip
WWW	World Wide Web

References

1. Jannach, D.; Felfernig, A.; Zanker, M.; Friedrich, G. *Recommender Systems. An Introduction*; Cambridge University Press: Cambridge, UK, 2011.
2. Bobadilla, J.; Ortega, F.; Hernando, A.; Gutiérrez, A. Recommender systems survey. *Knowl. Based Syst.* **2013**, *46*, 109–132. [CrossRef]
3. Adomavicius, G.; Tuzhilin, A. Context-aware recommender systems. In *Recommender Systems Handbook*; Springer: Berlin/Heidelberg, Germany, 2015; pp. 191–226.
4. Thai-Nghe, N.; Drumond, L.; Horvath, T.; Krohn-Grimberghe, A.; Nanopoulos, A.; Schmidt-Thieme, L. Factorization Techniques for Predicting Student Performance. In *Educational Recommender Systems and Technologies: Practices and Challenges*; IGI-Global: Hershey, PA, USA, 2012; pp. 129–153.
5. Adomavicius, G.; Tuzhilin, A. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Trans. Knowl. Data Eng.* **2005**, *17*, 734–749. [CrossRef]
6. Ricci, F.; Rokach, L.; Shapira, B. Introduction to recommender systems handbook. In *Recommender Systems Handbook*; Springer: Berlin/Heidelberg, Germany, 2011; pp. 1–35. [CrossRef]
7. Bobadilla, J.; Serradilla, F.; Bernal, J. A new collaborative filtering metric that improves the behavior of recommender systems. *Knowl. Based Syst.* **2010**, *23*, 520–528.
8. Herlocker, J.L.; Konstan, J.A.; Terveen, L.G.; Riedl, J.T. Evaluating collaborative filtering recommender systems. *ACM Trans. Inf. Syst. TOIS* **2004**, *22*, 5–53. [CrossRef]
9. Hernando, A.; Bobadilla, J.; Ortega, F. A non negative matrix factorization for collaborative filtering recommender systems based on a Bayesian probabilistic model. *Knowl. Based Syst.* **2016**, *97*, 188–202. [CrossRef]
10. Rendle, S.; Schmidt-Thieme, L. Online-updating regularized kernel matrix factorization models for large-scale recommender systems. In Proceedings of the 2008 ACM Conference on Recommender Systems, Lousanne, Switzerland, 23–25 October 2008; pp. 251–258. [CrossRef]
11. Unsalan, C.; Tar, B. *Digital System Design with FPGA: Implementation Using Verilog and VHDL*; McGraw-Hill: New York, NY, USA, 2017.
12. Tessier, R.; Pocek, K.; DeHon, A. Reconfigurable Computing Architectures. *Proc. IEEE* **2015**, *103*, 332–354.
13. Goeders, J.; Holland, G.M.; Shannon, L.; Wilton, S.J.E. Systems-on-Chip on FPGAs. In *FPGAs for Software Programmers*; Koch, D., Hannig, F., Ziener, D., Eds.; Springer International Publishing: Cham, Switzerland, 2016; pp. 261–283. doi:10.1007/978-3-319-26408-0_15. [CrossRef]
14. Vestias, M.; Neto, H. Trends of CPU, GPU and FPGA for high-performance computing. In Proceedings of the IEEE 24th International Conference on Field Programmable Logic and Applications, Munich, Germany, 2–4 September 2014; pp. 1–6. [CrossRef]
15. Terveen, L.; Hill, W.; Amento, B.; McDonald, D.; Creter, J. PHOAKS: A system for sharing recommendations. *Commun. ACM* **1997**, *40*, 59–62. [CrossRef]
16. Kautz, H.; Selman, B.; Shah, M. Referral Web: Combining Social Networks and Collaborative Filtering. *Commun. ACM* **1997**, *40*, 63–65. [CrossRef]
17. Balabanovic, M.; Shoham, Y. Fab: Content-Based, Collaborative Recommendation. *Commun. ACM* **1997**, *40*, 66–72. [CrossRef]
18. Pudhiyaveetil, A.K.; Gauch, S.; Luong, H.P.; Eno, J. Conceptual recommender system for CiteSeerX. In Proceedings of the 2009 ACM Conference on Recommender Systems, RecSys 2009. New York, NY, USA, 23–25 October 2009; Bergman, L.D., Tuzhilin, A., Burke, R.D., Felfernig, A., Schmidt-Thieme, L., Eds.; ACM: New York, NY, USA, 2009; pp. 241–244. [CrossRef]
19. Luettin, J.; Rothermel, S.; Andrew, M. Future of In-Vehicle Recommendation Systems @ Bosch. In *Proceedings of the 13th ACM Conference on Recommender Systems*; RecSys '19; Association for Computing Machinery: New York, NY, USA, 2019; p. 524. [CrossRef]
20. Ostuni, V.C. "Just Play Something Awesome": The Personalization Powering Voice Interactions at Pandora. In *Proceedings of the 13th ACM Conference on Recommender Systems*; RecSys '19; Association for Computing Machinery: New York, NY, USA, 2019; p. 523. [CrossRef]

21. Verma, R.; Ghosh, S.; Saketh, M.; Ganguly, N.; Mitra, B.; Chakraborty, S. Comfride: A Smartphone Based System for Comfortable Public Transport Recommendation. In *Proceedings of the 12th ACM Conference on Recommender Systems*; RecSys '18; Association for Computing Machinery: New York, NY, USA, 2018; p. 181–189. [CrossRef]
22. Tang, J.; Du, X.; He, X.; Yuan, F.; Tian, Q.; Chua, T. Adversarial Training Towards Robust Multimedia Recommender System. In *IEEE Transactions on Knowledge and Data Engineering*; IEEE: Piscataway, NJ, USA, 2019; pp. 1–1. [CrossRef]
23. Raja, D.R.K.; Pushpa, S. Diversifying personalized mobile multimedia application recommendations through the Latent Dirichlet Allocation and clustering optimization. *Multim. Tools Appl.* **2019**, *78*, 24047–24066. [CrossRef]
24. Amato, F.; Moscato, V.; Picariello, A.; Sperlì'ì, G. Extreme events management using multimedia social networks. *Future Gen. Comput. Syst.* **2019**, *94*, 444–452. [CrossRef]
25. Kuhl, N.; Lobana, J.; Meske, C. Do you comply with AI? —Personalized explanations of learning algorithms and their impact on employees' compliance behavior. *arXiv* **2020**, arXiv:2002.087772020. [CrossRef]
26. Meske, C.; Bunde, E. Using Explainable Artificial Intelligence to Increase Trust in Computer Vision. *arXiv preprint* **2020**, arXiv:cs.HC/2002.01543.
27. Pimenidis, E.; Polatidis, N.; Mouratidis, H. Mobile recommender systems: Identifying the major concepts. *J. Inf. Sci.* **2019**, *45*, 387–397.
28. Zhiwen Yu.; Xingshe Zhou.; Daqing Zhang.; Chung-Yau Chin.; Xiaohang Wang.; Ji Men. Supporting Context-Aware Media Recommendations for Smart Phones. *IEEE Pervas. Comput.* **2006**, *5*, 68–75. [CrossRef]
29. Lemos, F.; Carmo, R.; Viana, W.; Andrade, R. Towards a context-aware photo recommender system. *CEUR Workshop Proc.* **2012**, *889*. [CrossRef]
30. Sotsenko, A.; Jansen, M.; Milrad, M. Using a rich context model for a news recommender system for mobile users. *CEUR Workshop Proc.* **2014**, *1181*, 13–16.
31. Wang, X.; Rosenblum, D.; Wang, Y. Context-Aware Mobile Music Recommendation for Daily Activities. In *Proceedings of the 20th ACM International Conference on Multimedia*; MM '12; Association for Computing Machinery: New York, NY, USA, 2012; p. 99–108.
32. Baltrunas, L.; Ludwig, B.; Peer, S.; Ricci, F. Context relevance assessment and exploitation in mobile recommender systems. *Perso. Ubiquit. Comput.* **2012**, *16*, 507–526. [CrossRef]
33. Ma, Y.; Suda, N.; Cao, Y.; Vrudhula, S.; sun Seo, J. ALAMO: FPGA acceleration of deep learning algorithms with a modularized RTL compiler. *Integration* **2018**, *62*, 14–23. [CrossRef]
34. Gankidi, P.R.; Thangavelautham, J. FPGA architecture for deep learning and its application to planetary robotics. In Proceedings of the 2017 IEEE Aerospace Conference, Big Sky, MT, USA, 4–11 March 2017; pp. 1–9. [CrossRef]
35. Canilho, J.; Véstias, M.; Neto, H. Multi-core for K-means clustering on FPGA. In Proceedings of the 2016 26th International Conference on Field Programmable Logic and Applications (FPL), Lausanne, Switzerland, 29 August–2 September 2016; pp. 1–4. [CrossRef]
36. Winterstein, F.; Bayliss, S.; Constantinides, G.A. FPGA-based K-means clustering using tree-based data structures. In Proceedings of the 2013 23rd International Conference on Field programmable Logic and Applications, Porto, Portugal, 2–4 September 2013; pp. 1–6. [CrossRef]
37. Lin, Z.; Lo, C.; Chow, P. K-means implementation on FPGA for high-dimensional data using triangle inequality. In Proceedings of the 22nd International Conference on Field Programmable Logic and Applications (FPL), Oslo, Norway, 29–31 August 2012; pp. 437–442. [CrossRef]
38. Nagarajan, K.; Holland, B.; George, A.D.; Slatton, K.C.; Lam, H. Accelerating Machine-Learning Algorithms on FPGAs using Pattern-Based Decomposition. *J. Signal Proc. Syst.* **2011**, *62*, 43–63. [CrossRef]
39. Amazon EC2 F1 Instances. 2019. Available online: https://aws.amazon.com/cn/ec2/instance-types/f1/ (accessed on 22 February 2020). [CrossRef]
40. Bottou, L. Large-Scale Machine Learning with Stochastic Gradient Descent. In *Proceedings of 19th International Conference on Computational Statistics*; Springer: Heidelberg, Germany, 2010; pp. 177–186.
41. Kara, K.; Alistarh, D.; Alonso, G.; Mutlu, O.; Zhang, C. FPGA-Accelerated Dense Linear Machine Learning: A Precision-Convergence Trade-Off. In Proceedings of the 2017 IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), Napa, CA, USA, 30 April–2 May 2017; pp. 160–167.

42. Lee, D.D.; Seung, H.S. Unsupervised learning by convex and conic coding. In *Advances in Neural Information Processing Systems*; IEEE: Piscatawy, NJ, USA, 1997; pp. 515–521. [CrossRef]
43. Koren, Y.; Bell, R.; Volinsky, C. Matrix factorization techniques for recommender systems. *Computer* **2009**, *42*, 30–37.
44. Mnih, A.; Salakhutdinov, R.R. Probabilistic matrix factorization. In *Advances in Neural Information Processing Systems*; IEEE: Piscatawy, NJ, USA, 2008; pp. 1257–1264. [CrossRef]
45. Hoffman, M.D.; Blei, D.M.; Wang, C.; Paisley, J. Stochastic variational inference. *J. Mach. Learn. Res.* **2013**, *14*, 1303–1347.
46. Cong, J.; Liu, B.; Neuendorffer, S.; Noguera, J.; Vissers, K.; Zhang, Z. High-Level Synthesis for FPGAs: From Prototyping to Deployment. *IEEE Trans. Comput. Aided Des. Integr. Circ. Syst.* **2011**, *30*, 473–491.
47. Xilinx Inc. *Vivado Design Suite User Guide: High-Level Synthesis*; Technical Report UG902 (v2019.2); Xilinx Inc.: San Jose, CA, USA, 2020. [CrossRef]
48. O'Loughlin, D.; Coffey, A.; Callaly, F.; Lyons, D.; Morgan, F. Xilinx Vivado High Level Synthesis: Case studies. In Proceedings of the 25th IET Irish Signals Systems Conference 2014 and 2014 China-Ireland International Conference on Information and Communications Technologies (ISSC 2014/CIICT 2014), Limerick, Ireland, 26–27 June 2014; pp. 352–356.
49. Pajuelo-Holguera, F.; Gómez-Pulido, J.A.; Ortega, F.; Granado-Criado, J.M. Recommender system implementations for embedded collaborative filtering applications. *Microproce. Microsyst.* **2020**, *73*, 102997. [CrossRef]