On the Design of Scalable and Reusable Accelerators for Big Data Applications

(invited paper)

Christian Pilato, Qirui Xu, Paolo Mantovani,
Giuseppe Di Guglielmo, Luca P. Carloni
Dept. of Computer Science – Columbia University, New York, NY - USA
{pilato,paolo,giuseppe,luca}@cs.columbia.edu, qx2141@columbia.edu

ABSTRACT

Accelerators are becoming key elements of computing platforms for both data centers and mobile devices as they deliver energy-efficient high performance for key computational kernels. However, the design and integration of such components is complex, especially for Big Data applications where they have very large workloads to elaborate. Properly customizing the accelerators' private local memories (PLMs) is of critical importance. To analyze this problem we design an accelerator for Collaborative Filtering by applying a system-level design methodology that allows us to synthesize many alternative micro-architectures as we vary the PLM sizes. We then evaluate the resulting accelerators in terms of resource requirements for both embedded architectures and data centers as we vary the size and density of the workloads.

1. INTRODUCTION

Big Data applications are increasingly used to identify underlying structures and relations in very large unstructured data sets [2]. For example, the recommendation systems developed by companies like Amazon, Yelp, and Netflix, allow users to rate the items that they have purchased (e.g., books, restaurant meals, movies, etc.), through a predefined scale (e.g., one to five stars). This information is used to predict the ratings that the users would give to other items and make recommendations to other users with similar interests and profiles [4]. The machine-learning technique behind these systems, *Collaborative Filtering* [12], is finding increasing variety of applications [10, 25]. Collaborative-filtering algorithms, such as the Restricted Boltzmann Machine (RBM) [24], are computationally demanding. They typically consists of two main phases. During the training phase, an internal model is built through the analysis of very large data sets (e.g. hundreds of millions of ratings from hundreds of thousands of users [24]). Then, during the prediction phase, the model is used to make decisions such as determining which recommendations should be given to each user. Building the internal models can take a time of the order of days [15, 17], thereby degrading the performance of most recommendation systems. Additionally, the memory requirements keep increasing as the amount of data created by social media continues to grow together with user expectations.

Heterogeneous System-on-Chip (SoC) architectures [5] are increasingly used to design computing platforms for both mobile devices [11, 28] and data centers [13, 23]. To achieve energy-efficient high performance, these SoCs integrate many *specialized hardware accelerators*, which are equipped with *private local memories* (PLM) [9, 20]. PLMs offer predictable memory access latency and can be customized by increasing the number of concurrent memory operations to support higher parallelism [22]. The size of the data sets, however, is at least one order of magnitude larger than the accelerator PLM, whose range varies typically between tens and hundreds of kilobytes [9]. This gap is expected to grow further together with the amount of data to elaborate.

We present the design of an accelerator for the RBM algorithm as a case study to analyze the issues in the design of accelerators for Big Data applications. Such applications require the creation of *scalable* and *reusable* Intellectual Property (IP) blocks that scale efficiently with the problem size. This involves a careful co-design of the three main stages of accelerator operations: computation, communication, and storage. Reusable IPs allow the reduction of non-recurrent engineering costs [16], but their design and maintenance is estimated to be 25× more difficult than the design of an IP for one-time use [1]. A prerequisite to obtain highly reusable IPs is to raise the design entry-point from the register-transfer level (RTL) to *system-level design (SLD)* [6]. Describing the accelerators in a high-level description language such as SystemC allows designers to specify, parameterize, and analyze more easily these components. Then, high-level synthesis (HLS) tools can be used to automatically synthesize many alternative implementations from the same specification. This enables an effective design-space exploration, which enhances reusability [18].

After introducing the RBM algorithm and the general architecture of the RBM accelerator (Section 2), we discuss in detail the design of two IPs for the training and the prediction phases, respectively (Section 3). Our SLD methodology enables the exploration of different trade-off points in terms of resource requirements, PLM size, and characteristics of data transfers with DRAM (Section 4). In particular, it promotes the reuse of IP blocks across different SoC architectures with limited additional effort by the designer. Experimental results obtained with the implementation of a full-system prototype on an FPGA board show that the RBM accelerators achieve up to a 23× speed-up with respect to a software implementation running on an embedded processor (Section 5). These results also allow us to estimate that they would deliver a 300× improvement in terms of energy efficiency with respect to an Intel Sandybridge core when targeting an industrial 32nm CMOS technology for acceleration in data centers. Finally, we discuss how varying the size and density of the data sets to be processed impacts the performance of the accelerators.

CF'16 May 16-19, 2016, Como, Italy

ACM ISBN 978-1-4503-4128-8/16/05.

DOI: http://dx.doi.org/10.1145/2903150.2906141

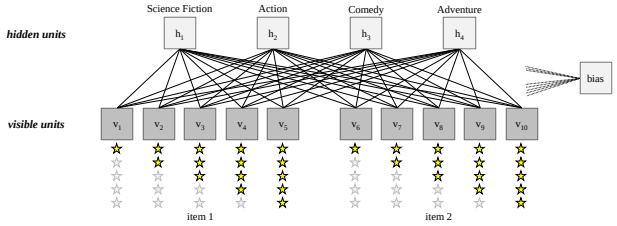


Figure 1: Model of the RBM: two-layer, bipartite neural network with connections between visible and hidden units for each rated item.

2. ACCELERATORS FOR BIG DATA

In this section we introduce the Restricted Boltzmann Machine (RBM) algorithm and present the high-level organization of our RBM accelerator.

Restricted Boltzmann Machine (RBM). This algorithm performs a binary version of factor analysis. Factor analysis describes the variability among observed, correlated variables, called *visible units*, in terms of a typically lower number of unobserved variables, called *hidden units* (or *latent factors*), which are unknown. An RBM is a *stochastic neural network* that has the structure of a bipartite graph: the two partitions of the network nodes (or neurons) correspond to the visible and hidden units, respectively. Each node has a binary state that can be either active (1) or non-active (0). Each visible unit is then connected to each hidden unit (and vice versa) through a pair of unidirectional edges. Further, each visible unit is connected to a bias unit that is always in active state and accounts for the inherent popularity of each movie item. The weights associated with these edges represent the model to be learned. In fact, a node activation state depends on those of its neighboring nodes, which influence it to a varying degree depending on the weight associated with the connecting edges. We apply RBM to a recommendation system that provides a user with movie suggestions based on a five-star rating scale. Fig. 1 shows a small portion of the RBM for this system, including ten visible units for two movie items (each movie has five associated units but exactly one is active, denoting its ratings in the five-star scale) and four hidden units (each associated with a different movie category such as "science fiction" or "comedy"). A complete RBM has a very large number of visible units (five times the number of movies in the database) and a much smaller number of hidden units [24].

Like many other machine-learning techniques, the RBM goes through a *training phase* before being ready to perform *predictions*, as shown by the control data flow graph (CDFG) of Fig. 2. The number *#movie* of movie items and the number F of latent factors are input parameters that determine the size of the network, along with the rating range K. Given a set of training users, the algorithm takes one input vector for each user, containing the movie ratings. Note that the average number of movies rated by the users is an indicator of the *density* of the data set.

The training algorithm takes the movie ratings from all the users as input data and tries to learn their relationships with respect to the movie categories (i.e. it tries to learn the latent factors represented by the hidden units). The graph in Fig. 2 shows that training data are first processed to determine the hidden-unit activation. From the latter, RBM derives the visible-unit states, which represent the suggestions made by the algorithm based on the weights of the connection edges. These suggestions are then compared with the training set to determine whether the network edges are weighted appropriately. This process repeats for a configurable number of iterations.

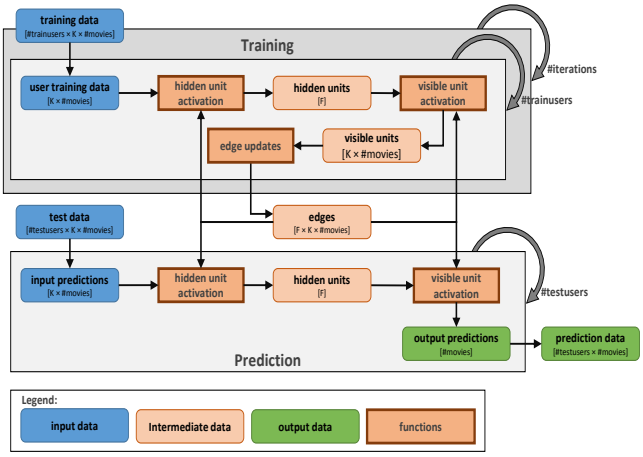
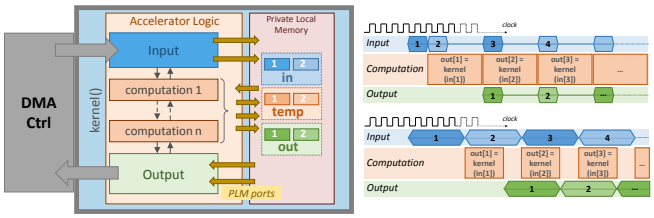


Figure 2: Control data flow graph of the RBM algorithm.

The result of the training phase is the model captured by the neural network and represented by the edge values. This is the only information shared between the two phases and is used to process a new set of users. In fact, differently from training, the prediction phase uses the model (i.e. the edge values) to activate the hidden and visible units based on partial users' data and actually perform the prediction on the remaining ones. In other words, test users have specified the ratings only for a subset of the movies, which are used to activate the hidden units. Based on such user preferences, this phase predicts the ratings for the remaining movies (i.e. the state of the corresponding visible units). In addition, no iteration is necessary to make such predictions because the network has been fixed during the previous phase.

The RBM algorithm is a representative case study for Big Data applications. In fact, both phases are data intensive and computationally complex. However, they usually have different requirements. The training phase requires the elaboration of very large training sets, usually for many iterations. Due to the size of these data sets, it demands high computational power and memory bandwidth. Thus the designers of training accelerators typically focus on obtaining high performance, even in exchange for larger area occupation or power dissipation. Conversely, accelerating the predictions on mobile devices aims at improving user's experience. In this case, the designers of prediction accelerators are likely to trade off some performance for more area and energy savings. Ideally, an accelerator should be designed in a way that can be easily optimized for integration in different platforms. Additionally, it is also important the co-optimization and verification of the data structures shared in memory.

Acceleration of Big Data Applications. Our approach to design heterogeneous SoC architectures is based on the loosely-coupled accelerator (LCA) model, which is particularly suited for realizing accelerators that process large data sets independently from the processor cores [9]. Being designed independently from the processors, an LCA is highly reusable and can be optimized to take advantage of a private local memory (PLM), which is carefully tailored to its specific needs. Application software can invoke and configure an LCA through a device driver that runs on top of the operating system (OS), which manages also the allocation in DRAM of the data that the accelerator must process. As shown in Fig. 3, an LCA autonomously perform *direct memory accesses* (DMA) to exchange data between the DRAM and their PLMs. Once the min-



(a) (b)

Figure 3: Accelerator's architecture (a) and two different execution-phase scenarios (b).

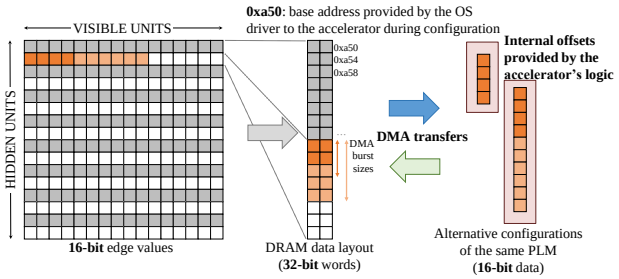


Figure 4: Data reorganization for improving communication with DRAM.

imum amount of data necessary for the computation is available in the PLM, the accelerator logic can start the elaboration and produce the corresponding part of results. It is important to note that, in case of Big Data applications, the DMA transfers are required not only to read consecutive parts of the input data set, but also to exchange temporary information (e.g. the model of the neural network) that cannot fit entirely in the PLM. In fact, in the RBM algorithm, the size of the neural network (i.e. the number of edge values) grows linearly with the number of movies. Hence, to design an accelerator that can work with any movie database it is necessary to store in DRAM the entire model and bring in and out only the portion of data necessary for each computation step.

Clearly, the size of the PLM has a significant impact on the accelerator performance and its DRAM communications. Specifically, customizing the PLM affects the three main aspects of the operations of any accelerator:

- **storage**: the PLM size determines how much data are stored locally to the accelerator at any given time and consequently its *memory cost* (e.g. silicon area or FPGA resources);
- **computation**: increasing the number of PLM banks that can be concurrently accessed enables the parallel execution of many memory operations, which is key to support micro-architectural optimizations (e.g. loop unrolling) [22];
- **communication**: the PLM size determines the number, frequency, and length of the DMA transfers.

Accelerators for Big Data applications require a concurrent optimization (at design time) and an efficient coordination (at run time) of these phases to avoid unexpected performance degradation. On the other hand, depending on the target scenario (e.g. data center or mobile device) or technology (e.g. standard cell-based design or FPGA device), the designer may choose different solutions.

3. ACCELERATOR DESIGN

In this section, we present the design of two IP blocks (TRAIN and PREDICT) to accelerate the training and prediction phases of the RBM algorithm. By using this design as a case study, we discuss how to realize accelerators for Big Data applications that highly scalable and reusable. This requires to optimize each of the three aspects discussed above, as well as their interactions.

Storage. In order to explore the PLM size, we parametrized the two accelerators with respect to the number of movies that can be concurrently processed. We used two different parameters, i.e. LM_t and LM_p, for the TRAIN and PREDICT accelerators, respectively. These parameters represent the number of movies to be processed for each iteration of the computational phases of our accelerators. The number of latent factors is instead a constant at design time (e.g. $F = 100$ [24]). This parameterization allows us to perform design-space exploration in different directions for the two accelerators, considering the different systems where they can be reused. The two parameters directly impacts the PLM size (storing more movies requires a larger PLM) and, consequently, resource requirements and power consumption. Notice that the PLM typically occupies a large portion of the accelerator area [9, 19].

Computation. Another important issue is the *density* of the data sets, defined as the number of relevant information in each chunk. For example, in the case of the RBM algorithm, we may have data sets storing only a small number of rating by each user. This is a common situation for Big Data applications and it is important to evaluate its impact on the computational phase. If the chunk of data under analysis does not contain much relevant information (e.g. the given user has rated very few movies), the subsequent procedures for activating hidden and visible units causes very few or no changes on the model (i.e. edge values). Hence, many DMA transfers to bring the model data in and out can be avoided. We made this optimization by analyzing the training values as soon as they are received. The subsequent DMA transfers are thus performed only for the relevant parts of the model. Additionally, we optimized the micro-architecture by tuning the precision of the fixed-point computation and designing optimized mathematical operators (e.g. sigmoid function).

Communication. From the communication viewpoint, the total number of movies (or corresponding visible units) must be decomposed in consecutive chunks to be processed separately. For this reason, the accelerators require to transfer potentially large chunks of consecutive data (i.e. set of visible units or edge values), depending on the available PLM size. The initiation of a DMA transfer has a small overhead that, however, accumulates during the execution. Hence, we must organize the data in DRAM in order to maximize the length of DMA transfers and minimize their number. As shown in Fig. 4, we reorganized the two-dimensional matrix of the edge values with respect to the reference implementation of the RBM algorithm. In this way, visible units associated with the same hidden unit are now stored consecutively in DRAM. The length of the DMA transfer is then proportional to the number of data that can fit in the given PLM. Additionally, the analysis of the needed fixed-point precision suggests that we can use less bits to represent the edge values than those offered by the given interconnection system. Hence, we chose a 16-bit representation and aggregated two consecutive values in the same 32-bit word, which is the size of our data line to main memory. This allows us to halve the length of the data transfers. In systems with larger data buses, it is possible to combine even more values in a single transfer.

Storage and Communication. The size of the PLM has a direct impact also on the communication between the accelerator and DRAM. For this reason, the matrix of edge values is partitioned so that it simplifies the load and store consecutive chunks, whose length is proportional to the PLM size (right-hand side of Fig. 4). In fact, the operation of starting a new DMA transfer takes only few cycles but is repeated thousands of times when processing large data sets with a small PLM.

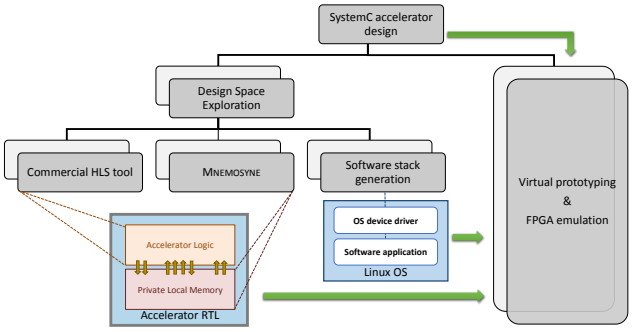


Figure 5: SLD methodology applied to the design and validation of accelerators for Big Data applications.

Storage and Computation. Increasing the PLM size also offers the possibility of exploiting more hardware parallelism on the data elaboration. For example, the accelerator logic can benefit from unrolling a loop to execute more operations in the same clock cycle at the cost of more resources. However, if the loop accesses a local data structure (e.g. part of the user's movie ratings) and the corresponding PLM is not properly optimized, this can impose a bottleneck on the computation. Hence, we used our PLM generator to create multi-bank architectures when needed [22]. This may further increase the resource requirements for the PLM. Indeed, when data are duplicated, it is necessary to add multiple replica of the same memory blocks into the accelerator. Instead, when the data are distributed across the different banks, we can generate smaller memory blocks. In this case, the total amount of memory is not increased, but composing a PLM with multiple smaller SRAMs requires more chip area than using a single but larger one (about 20% more in an industrial 32nm CMOS technology).

Communication and Computation. Overlapping computation and communication is an important optimization to improve the performance of applications that require large data transfers [9]. This can be obtained with ping-pong buffers at the cost of more PLM resources. However, when using this solution, the two phases have to be accurately balanced. In fact, increasing the length of the data transfers may limit the benefits obtained from the acceleration of the computational phase, as shown in Fig. 3. On one hand, a larger PLM size allows more hardware parallelism (potentially reducing the duration of the computation phase). On the other hand, longer data transfers need to be completed before the computation can start. As a result, if the computation is optimized so that it terminates before the next data chunk becomes available (i.e. the next data transfer is completed), there is no improvement for the application.

4. EXPERIMENTAL SETUP

We designed the TRAIN and PREDICT accelerators for the RBM application following the LCA model as discussed in Section 2. Starting from the reference C-based implementation provided in the CORTEXSUITE, a recently-released suite of benchmarks designed to challenge current hardware systems [26], we completed a design specification in synthesizable SystemC that is parametric with respect to the PLM size. We also created a model of the DMA controller in the testbench to evaluate the interactions with the DRAM, which is modeled in the testbench. Iwe evaluated how to organize its data structures in DRAM and how to co-optimize the data transfers of both TRAIN and PREDICT accelerators. This is especially important for the data structures shared between the two

Table 1: Characteristics of TRAIN and PREDICT accelerators for the different target technologies.

	PLM size	FPGA resources			CMOS 32nm		
		LUTs	FFs	BRAMs	Logic mm^2	PLM mm^2	Power mW
TRAIN	10	26,222	16,981	14	36,147	474,226	46.47
	20	27,284	17,018	18	37,049	798,394	76.11
	50	28,774	17,086	38	37,319	1,785,407	113.84
	100	31,953	17,174	70	38,487	3,101,740	181.82
PREDICT	10	22,152	12,988	5	42,305	89,369	26.60
	20	24,488	13,311	7	43,037	143,367	32.09
	50	26,921	13,755	15	44,954	308,315	38.82
	100	30,976	14,147	27	49,478	528,979	54.23

accelerators, i.e. the edge values. The SystemC specification can be simulated with a *virtual platform* to perform more complex analyses at the system level (e.g., on the interactions with the processor core and the OS) [3, 9]. System-level simulation enables a richer exploration of multiple design choices, (e.g., data-type precision) and a better analysis of concurrency aspects (e.g. communication and computation overlapping).

We then used an SLD methodology for the automatic generation of alternative implementations starting from the same high-level SystemC specification [20]. As shown in Fig. 5, the methodology is supported by a commercial HLS tool for generating the accelerator logic and by MNEMOSYNE, our prototype CAD tool for PLM customization [22]. This allows us to easily create multiple alternative designs (even for different technologies) and evaluate the impact of micro-architectural optimizations. Specifically, we synthesized the different implementations by varying the design parameters (e.g. PLM size) or target technology (e.g. ASIC or FPGA). We used the same SystemC-based testbench to verify the RTL netlist of the accelerators and their interactions with DRAM. Table 1 reports the characteristics of the RTL netlists for the TRAIN and PREDICT accelerators synthesized for both the FPGA and ASIC technologies. For the FPGA implementation, we performed logic synthesis with Xilinx Vivado 2015.2 with a target clock period of 80 MHz for a Xilinx Virtex-7 XC7V2000T. For the ASIC implementation, we performed logic synthesis with Synopsys Design Compiler with a target clock period of 1 GHz on an industrial 32nm CMOS technology library.

Our methodology greatly simplifies the subsequent design of the *software stack*, including the device driver to configure the accelerator and the software application required to prepare the data in memory. These steps can be validated with co-simulation, virtual prototyping or FPGA emulation [9, 20].

For evaluating the accelerators in a complete system, we created an FPGA-based prototype. This architecture features a Leon3 embedded processor core that runs a complete Linux OS to prepare the data in DRAM and configure the accelerators. We used two different training sets that are representative of Big Data application scenarios:

- DENSE: it is a modified version of the Netflix database and it is provided in CORTEXSUITE; it contains 89,031 ratings from 1,000 users on 100 movies (each user rated in average 89.03% of the movies).
- SPARSE: it contains movie rating from the MovieLens web site and it is provided by GroupLens Research[1]. It contains 100,000 ratings from 943 users on 1,682 movies (each user rated in average 6.30% of the movies).

[1] http://grouplens.org/datasets/movielens/

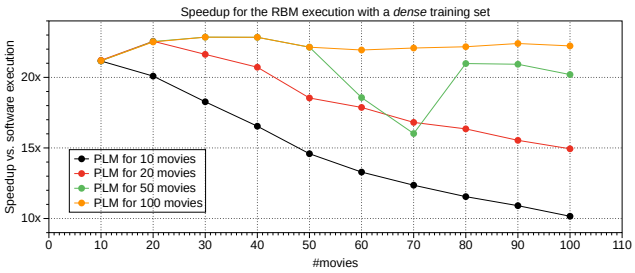


Figure 6: Speedup for the entire RBM application when processing a dense training set.

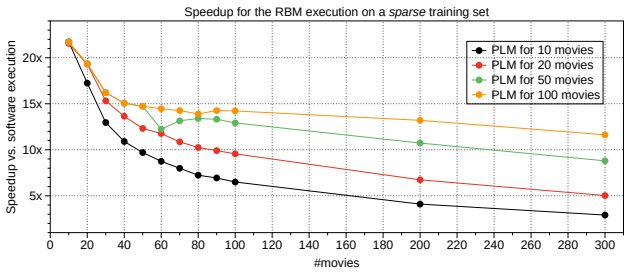


Figure 7: Speedup for the entire RBM application when processing a sparse training set.

5. EXPERIMENTAL ANALYSIS

In this section, we evaluate the design of the RBM accelerators while analyzing the impact of the PLM size, the characteristics of the data sets, and the interaction with DRAM.

Size of PLM. In the first set of experiments, we run the different versions of the accelerators in our FPGA-based prototype. In each experiment, we varied the size of the problem to elaborate (number of training users and movies) and the size of the accelerators PLM (the LM_t and LM_p parameters). We evaluated four configurations capable of storing locally 10, 20, 50, and 100 movies, respectively. Without overlapping communication and computation, we could achieve only a maximum speedup of 6× with respect to the software execution of the reference C-based implementation (from the CORTEXSUITE) running on the Leon3 core. Instead, the introduction of ping-pong buffers to overlap computation and communication yielded a maximum speedup of almost 23×. This was achieved with a large PLM ($LM_t = 100$ movies) when the size of the problem is large enough (more than 80 movies), as shown in Fig. 6. Moreover, in this situation, the speed-up is almost constant because the PLM is big enough to contain all the problem instances and no additional data transfers are required to store temporary data in DRAM. With smaller PLMs, however, the speed decreases as the problem size grows, due to the overhead of the additional communications with DRAM. It is worth notice that, when the number of movies is not a multiple of the PLM size (e.g. 60 or 70 movies with a PLM able to store only 50 movies), there is a negative impact on performance because there are computational iterations that are underutilized (i.e., the accelerators are stalling without elaborating any data as they are waiting for the next data chunk). This is clearly shown by the bend in the green curve of Fig. 6.

Density of data sets. We used the DENSE and SPARSE data sets to evaluate the impact of data density on accelerators' performance. We compared the results of executing the RBM application composed of the two combined TRAIN and PREDICT accelerators with the software execution on the Leon3 core. The resulting speed-ups are shown in Fig. 6 and Fig. 7 when varying the size of the PLM (for the sake of simplicity, LM_t is set equal to LM_p in all configurations). In these experiments, the speed-up is always decreasing as the problem increases, even if with different slopes. In fact, here the execution of the application is dominated by communications with the DRAM. The communication overhead is inversely proportional to the PLM size. In sparse data sets, it is frequent that any given user of the training set provides ratings only for a small subset of movies. This has different impacts for the operations of the accelerator and the processor. The accelerator operates at the granularity of the entire chunk, which is proportional to the PLM size. Larger PLMs bring in more data and the accelerator is thus more likely to have at least one element to elaborate. The processor, instead, brings in the cache line, which is usually much smaller in size than the PLM, only when the data is accessed. As a result, the optimization of skipping the computation for chunks with no data has limited impact (less than 1%) on the performance of the entire application.

Accelerators for custom chips. To prove the degree of reusability of our accelerator design, we assumed two different scenarios where the designers want to create a custom chip for accelerating the training in an embedded system and in a data center, respectively. For this analysis we used the ASIC implementation of the accelerators whose area and power consumption after logic synthesis are reported in Table 1. For the embedded domain, we designed an SoC with the Leon3 processor as described above, so that we could estimate its area occupation through logic synthesis with Synopsys Design Compiler. The processor requires roughly 0.5 mm^2, where data and instruction caches (16 KB each) are responsible for around 80% of its area. As shown in Table 1, the area of the RBM accelerators is even more dominated by memory in the case of an ASIC implementation. Given the requirements of embedded systems especially in terms of power consumption, a designer will likely use the smallest implementation of the TRAIN accelerator (i.e. with 10 movies in the PLM), which is 6× smaller than the largest one and consumes 4× less power, while it can deliver a 10× performance gain with respect to the processor.

For data centers, we combined the performance results (i.e. cycle counts) obtained though FPGA emulation with the power characterization to determine the energy consumption. We compared the results with the execution of the baseline C algorithm provided in the CORTEXSUITE on a server equipped with Intel Sandybridge E5-2430 chips (2.2 GHz), which are fabricated with the same technology node of our accelerator (i.e. 32nm) and have a die size of 294 mm^2 with a 64-KB L1 cache and a 256-KB L2 cache. To estimate the performance and power consumption, we instrumented the code with directives that access the processor's built-in power meters, as done in [14]. The results are reported in Table 2. When running a problem instance of 100 users and 100 movies for 200 training iterations, the Sandybridge processor completed the execution of the RBM application in 26.02 seconds with an average power consumption of 34.14 W. This corresponds to an energy consumption of 888.36 J. While operating at 1 GHz due to limitations in the available SRAM technology, our accelerators had a performance comparable to the server processor, with a fraction of the power consumption. For example, the design with a PLM capable of storing 100 movies was 1.7× faster than Sandybridge with only 0.35% of its energy consumption. These results are comparable with the ones obtained with other similar accelerators for Big Data applications (e.g. [7, 29]).

Table 2: Energy efficiency of the TRAIN accelerator compared to Intel Sandybridge.

	Conf. PLM	Power (mW)	Exec. time (s)	Energy (J)	Die Area (mm^2)
Intel Sandybridge	-	34,140.79	26.02	888.36	294.00
TRAIN	10	46.47	33.80	1.57	0.51
	20	76.10	22.98	1.75	0.84
	50	113.84	17.01	1.94	1.82
	100	181.82	15.50	2.81	3.14

6. RELATED WORK

Multi-machine architectures have been proposed to accelerate the training phase of machine learning algorithms [8]. Previous works in the literature have shown that specialized hardware accelerators for machine-learning applications can deliver high performance at a fraction of the energy cost of software execution [7, 17, 30]. Chen et al. [7] proposed an efficient accelerator for machine learning with an extensive analysis of data-type conversion and memory transfers. Ly and Chow [17] presented a hardware architecture for RBM while focusing on the exploitation of multiple FPGAs to scale the neural network. Zhang et al. [30] used C-based HLS to design an accelerator for Deep Convolutional Neural Networks that is highly optimized for FPGA; in doing so, they focused on the memory aspects of the problem, rather than creating reusable implementations. Differently from all these works, we focused on developing a design methodology that enables the automatic synthesis of many alternative implementations, thus enhancing the reusability of the accelerator.

Trinh et al. [27] worked on data representation problems, but their analysis is applied to the RTL implementation of the neural networks. In contrast, we promote the use of SLD and high-level programming languages, like SystemC, to validate these aspects at early stages of the design process. Analysis of fixed-point computation has been also addressed in the context of approximate computing to improve resource usage at the cost of a tolerated error in the results [21]. Similar approaches can be easily integrated and validated at the system level with our methodology.

7. CONCLUDING REMARKS

We discussed the problem of designing scalable and reusable accelerators for Big Data applications by using the two computationally intensive parts of the RBM algorithm, namely the training and prediction phases, as a case study. We used a SLD methodology to synthesize many alternative implementations of these accelerators from the same SystemC description. We analyzed how key design choices impact the performance of the resulting accelerated RBM application for different scenarios. The accelerator implementations generally scale well with the size of the problem and deliver significant gain in performance (up to 23× for an embedded system prototyped on FPGA) and energy (up to 300× for an ASIC implementation targeting data centers). On the other hand, we showed that the performance of accelerators for Big Data applications is also highly dependent on the characteristics of the training data sets. This aspect is becoming more and more critical due the constantly growing amount of data produced by social media and Internet services. Future work will be focused on the optimization of the data transfers with DRAM, especially in the case of sparse data sets.

Acknowledgments. The authors would like to thank Martha Kim (Columbia University) for her support with power measurements on Intel Sandybridge. This work is partially supported by C-FAR, one of the six centers of STARnet, the DARPA PERFECT program (Contract No: HR0011-13-C-0003), and the NSF (A#:1219001).

8. REFERENCES

[1] International Technology Roadmap for Semiconductors - 2011. Available at http://public.itrs.net.

[2] The data deluge: Challenges and opportunities of unlimited data in statistical signal processing. In *Proc. of ICASSP* (Apr. 2009), pp. 3701–3704.

[3] AARNO, D., AND ENGBLOM, J. *Software and System Development using Virtual Platforms*. Morgan Kaufmann, 2014.

[4] ADOMAVICIUS, G., AND TUZHILIN, A. Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering 17*, 6 (June 2005), pp. 734–749.

[5] BORKAR, S., AND CHIEN, A. A. The future of microprocessors. *Communication of the ACM 54* (May 2011), pp. 67–77.

[6] CARLONI, L. P. From latency-insensitive design to communication-based system-level design. *Proc. of the IEEE 103*, 11 (Nov. 2015), 2133–2151.

[7] CHEN, T., ET AL. DianNao: A Small-footprint High-throughput Accelerator for Ubiquitous Machine-learning. In *Proc. of ASPLOS* (Feb. 2014), pp. 269–284.

[8] CHILIMBI, T., ET AL. Project Adam: Building an Efficient and Scalable Deep Learning Training System. In *Proc. of OSDI* (Oct. 2014), pp. 571–582.

[9] COTA, E., ET AL. An analysis of accelerator coupling in heterogeneous architectures. In *Proc. of DAC* (June 2015), pp. 1–6.

[10] DABOV, K., ET AL. Image Denoising by Sparse 3-D Transform-Domain Collaborative Filtering. *IEEE Trans. on Image Processing 16*, 8 (Aug 2007), 2080–2095.

[11] GOULDING-HOTTA, N., ET AL. The GreenDroid Mobile Application Processor: An Architecture for Silicon's Dark Future. *IEEE Micro 31*, 2 (March 2011), 86–95.

[12] HERLOCKER, J., ET AL. Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems 22*, 1 (Jan. 2004), pp. 5–53.

[13] JOHNSON, C. L., ET AL. A Wire-Speed Power^{TM} processor: 2.3GHz 45nm SOI with 16 cores and 64 threads. In *Proc. of ISSCC* (2010), pp. 104–105.

[14] KAMBADUR, M., AND KIM, M. An experimental survey of energy management across the stack. In *Proc. of OOPSLA* (Oct. 2014), pp. 329–344.

[15] KAUTZ, H., ET AL. Referral web: Combining social networks and collaborative filtering. *Communication of the ACM 40*, 3 (Mar. 1997), pp. 63–65.

[16] KEATING, M., AND BRICAUD, P. *Reuse Methodology Manual for System-on-a-Chip Designs*, 3rd ed. Springer Publishing Company Inc., 2007.

[17] LE LY, D., AND CHOW, P. High-performance reconfigurable hardware architecture for restricted Boltzmann machines. *IEEE Trans. on Neural Networks 21*, 11 (Nov. 2010), pp. 1780–1792.

[18] LIU, H.-Y., PETRACCA, M., AND CARLONI, L. P. Compositional system-level design exploration with planning of high-level synthesis. In *Proc. of DATE* (Mar. 2012), pp. 641–646.

[19] LYONS, M., ET AL. The accelerator store: A shared memory framework for accelerator-based systems. *ACM Trans. on Arch. and Code Opt. 8*, 4 (Jan. 2012), pp. 1–22.

[20] MANTOVANI, P., ET AL. High-Level Synthesis of Accelerators in Embedded Scalable Platforms. In *Proc. of ASPDAC* (Jan. 2016), pp. 1–8.

[21] MOREAU, T., ET AL. SNNAP: Approximate computing on programmable SoCs via neural accelerator. In *Proc. of HPCA* (Feb. 2015), pp. 603–614.

[22] PILATO, C., ET AL. System-level memory optimization for high-level synthesis of component-based SoCs. In *Proc. of CODES+ISSS* (Oct. 2014), pp. 1–10.

[23] PUTNAM, A., ET AL. A reconfigurable fabric for accelerating large-scale datacenter services. In *Proc. of ISCA* (June 2014), pp. 13–24.

[24] SALAKHUTDINOV, R., ET AL. Restricted Boltzmann machines for collaborative filtering. In *Proc. of ICML* (2007), pp. 791–798.

[25] TAYLOR, G., ET AL. Modeling human motion using binary latent variables. *Advances in Neural Information Processing System 19* (2004), pp. 1345–1352.

[26] THOMAS, S., ET AL. CortexSuite: A synthetic brain benchmark suite. In *Proc. of IISWC* (Oct 2014), pp. 76–79.

[27] TRINH, H.-P., ET AL. Efficient data encoding for convolutional neural network application. *ACM Trans. on Arch. and Code Opt. 11*, 4 (Jan. 2015), pp. 49:1–49:21.

[28] WANG, A., ET AL. Heterogeneous multi-processing quad-core CPU and dual-GPU design for optimal performance, power, and thermal tradeoffs in a 28nm mobile application processor. In *Proc. of ISSCC* (Feb. 2014), pp. 180–181.

[29] WU, L., ET AL. Q100: The architecture and design of a database processing unit. In *Proc. of ASPLOS* (Feb. 2014), pp. 255–268.

[30] ZHANG, C., ET AL. Optimizing FPGA-based accelerator design for deep convolutional neural networks. In *Proc. of FPGA* (Feb. 2015), pp. 161–170.