This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMSCS.2018.2854787, IEEE Transactions on Multi-Scale Computing Systems

Hardware Accelerated Mappers for Hadoop MapReduce Streaming

Katayoun Neshatpour, Maria Malik, Avesta Sasan, Setareh Rafatirad, Houman Homayoun

Abstract—Heterogeneous architectures have emerged as an effective solution to address the energy-efficiency challenges. This is particularly happening in data centers where the integration of FPGA hardware accelerators with general purpose processors such as big Xeon or little Atom cores introduces enormous opportunities to address the power, scalability and energy-efficiency challenges of processing emerging applications, in particular in domain of big data. Therefore, the rise of hardware accelerators in data centers, raises several important research questions: What is the potential for hardware acceleration in MapReduce, a defacto standard for big data analytics? What is the role of processor after acceleration; whether big or little core is most suited to run big data applications post hardware acceleration? This paper answers these questions through methodical real-system experiments on state-of-the-art hardware acceleration platforms. We first present the implementation of four highly used big data applications in a heterogeneous CPU+FPGA architecture. We develop the MapReduce implementation of K-means, K nearest neighbor, support vector machine and naive Bayes in a Hadoop Streaming environment that allows developing mapper functions in a non-Java based language suited for interfacing with FPGA based hardware accelerating environment. We present a full implementation of the HW+SW mappers on existing FPGA+core platform and evaluate how a cluster of CPUs equipped with FPGAs uses the accelerated mapper to enhance the overall performance of MapReduce. Moreover, we study how various parameters at the application, system and architecture levels affect the performance and power-efficiency benefits of Hadoop streaming hardware acceleration. This analysis helps to better understand how presence of HW accelerators for Hadoop MapReduce, changes the choice of CPU, tuning optimization parameters, and scheduling decisions for performance and energy-efficiency improvement. The results show a promising speedup as well as energy-efficiency gains of upto 5.7× and 16× is achieved, respectively, in an end-to-end Hadoop implementation using a semi-automated HLS framework. Results suggest that HW+SW acceleration yields significantly higher speedup on little cores, reducing the performance gap between little and big cores after the acceleration. On the other hand, the energy-efficiency benefit of HW+SW acceleration is higher on the big cores, which reduces the energy-efficiency gap between little and big cores. Overall, the experimental results show that a low cost embedded FPGA platform, programmed using a semi-automated HW+SW co-design methodology, brings significant performance and energy-efficiency gains for Hadoop MapReduce computing in cloud-based architectures and significantly reduces the reliance on large number of big high-performance cores.

Index Terms—FPGA acceleration, hardware+software co-design, MapReduce, Hadoop streaming, Big-little core

1 Introduction

Emerging big data analytics applications require a significant amount of server computational power. However, these applications share many inherent characteristics that are fundamentally different from traditional desktop, parallel, and scale-out applications [3]. They heavily rely on specific deep machine learning and data mining algorithms. The characteristics of big data applications necessitates a change in the direction of server-class microarchitecture to improve their computational efficiency. However, while demand for data center computational resources continues to grow with the growth in the size of data, the semiconductor industry has reached scaling limits and is no longer able to reduce power consumption in new chips. Thus, current server designs based on commodity homogeneous processors, are no longer efficient in terms of performance/watt to process big data applications [4].

To address the energy efficiency problem, heterogeneous architectures have emerged to allow each application to run on a core that best matches its resource needs than a one size-fits-all processing node. In big data domain, various frameworks have been developed that allow the processing of large data sets with parallel and distributed algorithms. MapReduce [5] is an example of such frameworks developed by Google, which achieves high scalability and fault-tolerance for a variety of applications. While hardware acceleration has been applied to software implementation of widely used applications, MapReduce implementation of such applications requires new techniques, which studies their MapReduce implementation and their bottlenecks, and selects the most efficient functions for acceleration [6].

The rise of hardware accelerators in data centers, raises several important research questions: what is the potential for hardware acceleration in MapReduce, a defacto standard for big data analytics? How much performance benefits a semi-automated high level synthesis framework which is used for conventional compute-intensive applications bring for accelerating big data analytics applications? What is the role of processor after acceleration; whether big or little core is most suited to run big data application post hardware acceleration? How tuning optimization parameters at system, architecture and application levels for performance and energy-efficiency improvement changes before and after hardware acceleration? and how presence of hardware accelerator changes the scheduling decision for performance and energy-efficiency optimization? This paper answers all above questions through methodical real-system experiments on state-of-the-art hardware acceleration platforms.

To understand the potential performance gain of using a semi-automated standard HW+SW co-design methodology to accelerate analytics applications in MapReduce environment, in this paper we present a MapReduce implementa-

- *Department of Electrical and Computer Engineering, George Mason University, Fairfax, VA, 22030.*
E-mail: see kneshatp, mmalik9, asasan, srafatir hhomayou@gmu.edu

This paper has been presented in part as a poster in CCGRID 2015 [1] and as an invited paper in CODES-ISSS 2016 [2].

2332-7766 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMSCS.2018.2854787, IEEE Transactions on Multi-Scale Computing Systems

tion of big data analytics applications on a 12-node server MapReduce and evaluate a heterogeneous CPU+FPGA architecture, taking into account various communication and computation overhead in the system including the communication overhead with FPGA. We offload the hotspot functions in the mapper to the FPGA. We measured power, and execution time on the server and the FPGA board. To the best of our knowledge this is the first empirical work that focuses on the acceleration of Hadoop streaming for non-Java based map and reduce functions to find architectural insights. For evaluation purposes, we are performing the following tasks in this paper:

- MapReduce parallel implementation of various data mining and machine-learning application in C through Hadoop streaming.
- Implementation of HW+SW co-design of the mapper functions on the FPGA+core platform.
- Evaluating the overall speedup, power and energy-efficiency of the system considering various hardware communication and software computation overhead in Hadoop MapReduce environment.
- Performance and energy-efficiency analysis of the hardware acceleration based on application (size of input data), system (number of mappers running simultaneously per node and data split size), and architecture (big vs little core) level parameters.

Consequently, we make the following major observations:

- The optimal application, architecture, and system-level parameters to maximize the performance and energy-efficiency is different before and after acceleration.
- HW+SW acceleration yields higher speedup on little Atom cores, therefore significantly reducing the performance gap between little and big cores after acceleration.
- HW+SW acceleration yields higher energy-efficiency improvement on big Xeon cores, therefore significantly reducing the energy-efficiency gap between little and big cores.
- In presence of hardware acceleration, we can reduce the number of mapper/reducer slots (active cores) and yet be as energy-efficient as a case in which, all the available cores are active without significant performance loss.
- HW+SW acceleration improves scheduling decisions and provides more opportunities for fine-tuning of parameters to further enhance performance.

The rest of the paper is organized as follows: In Sec. 2 we provide a background on MapReduce. In Sec. 3 and 4 the system architecture and methodology are described, respectively. Sec. 6 introduces the studied big data applications. Sec. 5 describes the model for the calculation of the execution times after the acceleration. Sec. 7 describes the HW+SW co-design of the mapper functions. In Sec. 9 we show the results and carry out a sensitivity analysis on different configurations. In Sec. 10 and 11, we study scheduling of various workloads and the scalability of our framework, respectively. In Sec. 12, we discuss the related work. Finally, Sec. 13 concludes the paper.

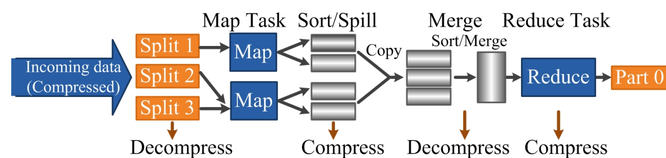


Fig. 1. Hadoop MapReduce: Computational Framework Phases [7].

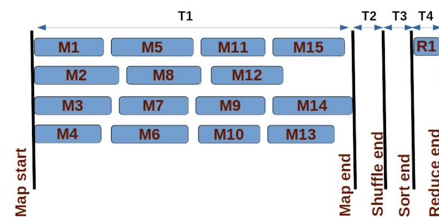


Fig. 2. Timing of various Hadoop phases

2 Hadoop MapReduce

MapReduce is the programming model developed by Google to handle large-scale data analysis. Fig. 1 shows the various phases in the MapReduce platform. The map functions parcel out the work to different nodes in the distributed cluster. They process <key/value> pairs to generate a set of intermediate <key/value> pairs. The reduce functions merge all the intermediate values with the same intermediate key and collate the work to resolve the results.

Apache Hadoop is an open-source Java-based framework of MapReduce implementation. It assists the processing of large datasets in a distributed computing environment and stores data in highly fault-tolerant distributed file system (HDFS).

2.1 Timing

Fig. 2 shows the timing diagram of a MapReduce application with 15 map jobs, one reduce job and four slots. A slot is a map/reduce computation resource unit at a node. The map phase starts with the start of the first map task and finishes when the last map task completes its execution. Shuffle starts shortly after the first map, and will not complete until all the map tasks are finished. A low-volume shuffle is finished shortly after the last map (e.g. Fig. 2), while a high-volume shuffle takes longer to complete. The sort phase finishes after the shuffle. Reduce starts after all the data is sorted. Upon the completion of all the reduce tasks, the whole MapReduce job is finished.

In the MapReduce platform, a significant portion of the execution time is devoted to the map and reduce phase, as they carry out the computation part. In this paper, we target the map phase for acceleration, as it accounts for a higher portion of the execution time across studied machine learning kernels.

3 System Architecture

In a general-purpose CPU several identical cores are connected to each other through their shared-memory distributed interconnect. However, for hardware acceleration, each core is extended with a small FPGA fabric. We study how adding on-chip FPGAs to each core would enhance the

2332-7766 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMSCS.2018.2854787, IEEE Transactions on Multi-Scale Computing Systems

TABLE 1
Architectural parameters

Processor	Intel Atom C2758	Intel Xeon E5-2420	Intel Xeon E5-2670
Cores\Threads	8\8	6\12	8\16
Operating Frequency	2.4 GHz	1.9 GHz	2.6 GHz
Micro-architecture	Silvermont	Sandy Bridge	Sandy Bridge
L1i Cache	32 KB	32 KB	32 KB
L1d Cache	24 KB	32 KB	32 KB
L2 Cache	4 MB	256 KB	256 KB
L3 Cache	-	15MB	20MB
System Memory	8 GB	32 GB	365 GB
TDP	20 W	95 W	115 W

performance of the architecture that runs Hadoop MapReduce. Fig. 3 shows the system architecture of the proposed multi-node platform studied in this paper. The single-node architecture is identical to the DataNode.

3.1 Single-node

While in a general purpose CPU, mapper/reducer slots are mapped to a single core, in the heterogeneous architecture depicted in Fig. 3, each mapper/reducer slot is mapped to a core that is integrated with the FPGA. Given the tight integration between FPGA and CPU, the interconnection interface between the two is the main overhead in this architecture. Thus, the mapper/reducer slots are accelerated with the FPGA, without any high off-chip data transfer overhead.

For implementation purposes, we compare two types of core architectures; little core Intel Atom C2758, and big core itel Xeon E5-2420. These two types of servers represent two schools of thought in server architecture design: using big Xeon cores, which is a conventional approach to designing a high-performance server, and Atom, which uses low-power cores to address the dark silicon challenge facing servers [8]. Table 1 shows the details of the studied servers.

Moreover, each FPGA in Fig. 3 is a low cost Xilinx Artix-7 with 85 KB logic cells and 560 KB block RAM. The integration between the core and the FPGA is compatible with the advanced micro-controller bus architecture (AMBA).

Specifically, we utilize measurements for the Advanced eXtensible Interface (AXI)-interconnect. AXI is an interface standard through which, different components communicate with each other. The data transferred between the core and the FPGA, is rearranged to create transfer streams. A direct memory access (DMA) engine is used to move streams in and out of the shared memory between the FPGA and the core, which provides high-bandwidth direct memory access between the AXI-stream and the IP interfaces implemented on the FPGA.

3.2 Multi-node

The architecture of the multi-node cluster consists of a homogeneous CPU as the NameNode, which is connected to several DataNodes with heterogeneous architectures. The architecture of each DataNode is similar to that in Fig. 3.

The NameNode is responsible for the job scheduling between all the DataNodes. It is configured to distribute the computation workloads among the DataNodes. The number of mapper/reducer slots on each DataNode is based on its number of cores. The interconnection between the NameNode and DataNodes is established through a multi

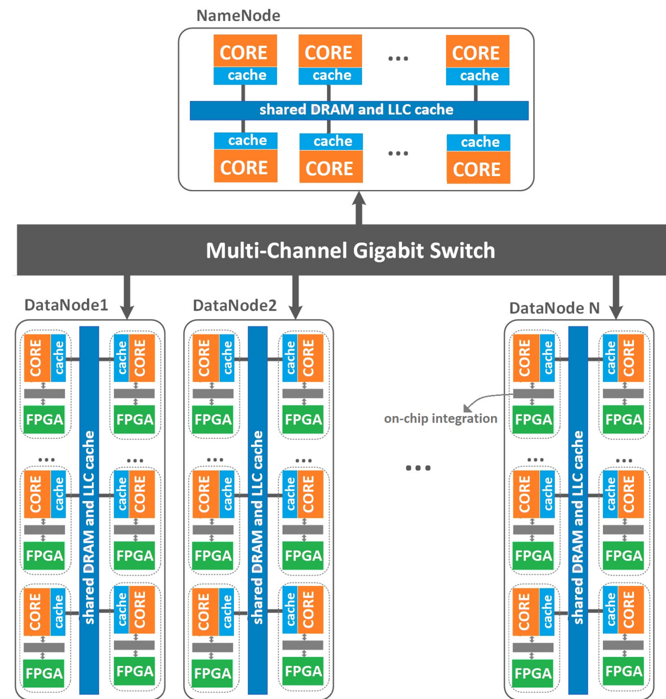


Fig. 3. System architecture for a multi-node cluster.

channel Gigabit switch to allow high data transfer rates. For implementation purposes, we use a 12-node cluster with E5-2670 CPUs. Table 1 shows the specifications of the E5-2670 CPUs.

4 Methodology

We develop a MapReduce version of each studied machine learning application for execution on Apache Hadoop. It should be noted that Hadoop applications are mostly implemented in Java; however, the FPGA+CPU platforms allow hardware acceleration of C-based applications. While native C/C++ injection into Hadoop is on the way, various utilities have been used to allow Hadoop to run applications, the map and reduce functions of which, are developed in languages other than Java. Hadoop pipes [9] and Hadoop streaming [10] are examples of such utilities. In this paper, we use Hadoop streaming, a utility that comes with the standard Hadoop distribution. It allows running MapReduce jobs with any executable or script as the mapper and/or the reducer. The Hadoop streaming utility creates a MapReduce job and submits it to an appropriate cluster.

This paper aims to characterize data mining and machine learning applications. Accordingly, the C-based implementation of such applications were developed and executed on Apache Hadoop streaming. Subsequently, two levels of profiling is carried out.

subsectionProfiling of the Application on MapReduce As mentioned earlier, the MapReduce platform is comprised of several execution phases; however, in this study we focus on the acceleration of the map phase. In order to calculate the potential speedup on the Hadoop platform after the acceleration of map functions for each application, we need a detailed analysis and profiling for various phases.(i.e. map, reduce, shuffle, etc.) We use the timing information

2332-7766 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMSCS.2018.2854787, IEEE Transactions on Multi-Scale Computing Systems

to calculate the execution time of each phase before the acceleration.

subsectionProfiling of the map function To accelerate the map functions through HW+SW co-design, we profile the map functions in order to find out the execution time of different sub-functions and select which functions should be offloaded to the hardware in the FPGA, and which ones still need to remain in the software on the core (i.e., SW part).

The map function is carried out on data splits. The data split size is mostly the size of one HDFS block for data locality purposes, which varies from 64MB, 128MB to higher values [11]. For each application, we execute the map function on the data splits and profile it on the two big and little server architectures. Profiling of map functions on these two architectures determine the sub-functions better suited for FPGA implementation.

The execution time of the accelerated map function is comprised of three parts. First, the SW part of the map function that remains on the core ($t_{sw,Atom}$ and $t_{sw,xeon}$ on Atom and Xeon, respectively), which is calculated based on the profiling of map functions on Intel Xeon and Atom using the Perf tool. Second, the HW part of the map function that is offloaded to the FPGA (t_{hw}), which is calculated by measurements from the FPGA implementations. And third, the data transfer time between the FPGA and the core (t_{tr}). The calculation of the transfer time requires a platforms that allows the integration of the FPGA with the core.

In the proposed framework the time-intensive sub-function within the map functions are targeted for acceleration. The memory patterns for map functions is highly regular with low data dependencies. Thus, profiling and measurements of execution time using gprof for map functions takes into account the time to bring the data from the main memory. In the accelerated map function, the selected sub-function is replaced with an accelerator, its execution time is replaced with the processing time of the accelerator and the time to stream the data between the map sub-functions on CPU to accelerated map sub-functions on FPGA.

In lack thereof a variety of high performance CPU+ on-chip FPGA platforms, to demonstrate how on-chip integration of the CPU and FPGA allows accelerating of sub-functions within Map functions and subsequently accelerates the overall MapReduce applications, rather than using raw and optimistic values reported for AXI-interconnection delay and bandwidth, we use timer functions on Zynq to calculate the timing of transmission.

To estimate the execution time after the acceleration accurately, and for functional verification, we fully implement the map functions on the Zedboard. ZedBoard (featuring XC7Z020 Zynq) integrates two 667 MHz ARM Cortex-A9 with an Artix-7 FPGA with 85 KB logic cells and 560 KB block RAM. The connections between the core and FPGA is established through the AXI interconnect [12]. To calculate the time required to send the data for each map function to the accelerator and the processing time of the FPGA, we add timer IPs on the FPGA. Using the timer, we measure the data transfer time (t_{tr}), and the accelerator time t_{hw}. The measurements are used as estimation for a framework, in which the transmission link and the FPGA are identical to those used in the Zynq platform, while the CPU is Intel Atom or Intel Xeon.

Based on the timings calculated from the full implementations on the Artrix-7 FPGA in this platform, the execution time of the map function after acceleration is calculated as $t_{sw,xeon} + t_{hw} + t_{tr}$ and $t_{sw,Atom} + t_{hw} + t_{tr}$, for Xeon and Atom, respectively, assuming a tight on-chip integration of the Intel Atom and Xeon cores to the studied FPGA. We compare the execution time of the map function before and after the acceleration to yield the speedup of the map functions though HW+SW co-design.

Finally, based on the information about the execution time of each phase, and the speedup of the map functions, we perform a comprehensive analysis to find out how acceleration of the map function contributes to the acceleration of the entire application on Hadoop MapReduce.

5 Estimating Hadoop MapReduce Speedup

In order to calculate the potential speedup on the Hadoop platform after the acceleration of map functions, we perform the following two steps: In the first step, the speedup of the map function through HW+SW co-design is calculated. In the second step, the speedup of the overall MapReduce is calculated, when the map function is accelerated with the rate calculated in the first step. As mentioned earlier, given the tight integration of FPGA and CPU, the main overhead is the on-chip data transfer between the core and the FPGA, which is calculated using the timers implemented on the Zynq platform.

5.1 Modelling Speedup of the Map Function through HW+SW co-design

A comparison of various models of computation for hardsware+software co-design has been presented by [13]. The classical FSM representation or various extension of it are the most well-known models for describing control systems, which consists of a set of states, inputs, outputs and a function which defines the outputs in terms of inputs and states, and a next-state function. Since they do not allow the concurrency of states and due to the exponential growth of the number of their states as the system complexity rises, they are not the optimal solution for modeling HW+SW co-design. Dataflow graphs have been quite popular in modeling data-dominated systems [14]. In such modeling, computationally intensive systems and/or considerable transportation of data is conveniently represented by a directed graph where the nodes describe computations and the arcs represent the order in which the computations are performed.

In case of acceleration of the map phase in a MapReduce platform, what needs to be taken into account is the highly parallel nature of the map functions, which allows higher acceleration by concurrent processing of multiple computations that have no data dependencies. Most efforts for modeling of the hardware+software co-design have found data dependencies to be an important barrier in the extent to which a function is accelerated, however this is not the case for MapReduce. In the mapper part of most machine-learning applications a small function, i.e., an inner product or a Euclidean distance calculation is the most time-consuming part of the code, where multiple instances of a small function can be executed in parallel with no data dependencies. In such cases, a simple queuing network

2332-7766 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMSCS.2018.2854787, IEEE Transactions on Multi-Scale Computing Systems

can be deployed to model a map function, with only one accelerated sub-function.

Queuing system models are useful for analyzing systems where inputs arrive sporadically or the processing time for a request may vary [14]. In a queuing model, customers (in this case, the data to be processed by the accelerator) arrive at the queue at some rate; the customer at the head of the queue is immediately taken by the processing node (the accelerator hardware), but the amount of time spent by the customer in processing must be specified (service time). Typically, both the customer arrival rate and processing time are modeled as Poisson random variables. In our case, however, since we are using one or multiple copies of the same accelerator, the service time for all data is fixed and is determined by the maximum frequency of the accelerator on the FPGA and the number of clock cycles it takes to finish the processing of each batch of data. Moreover, we assume that the data arrives at the accelerator at a fixed rate, determined by the processing speed of the accelerator.

Let T_{map} be the execution time of the map function on the data splits before the acceleration. Accordingly, based on the discussion in Sec. 4, the speedup of the map function is calculated as follows.

$$S_{map} = \frac{T_{map}}{t_{sw} + t_{hw} + t_{tr}}, \quad (1)$$

where t_{sw} is the SW part of the map function that remains on the core ($t_{sw,Atom}$ and $t_{sw,xeon}$ on Atom and Xeon, respectively), $t_{tr} + t_{hw}$ is HW time and the transfer time derived from the timers in Zynq implementation, and S_{map} is the speedup of the map function.

To calculate t_{hw}, we fully optimize the accelerator IPs using Vivado HLS. Vivado HLS reports the delay of the IPs in terms of the number of cycles. Based on the frequency of the accelerator in the implementations, the processing time of the IPs is measured (t_{hw}). To calculate t_{tr}, timer IPs are added on the FPGA. In the software codes, the timer is reset when a transfer is started to the PL. The timer is stopped when the Tlast is high, which happens after the last packet is processed by the PL. Thus, the timer indicates the time required for transfer of data through DMA and the processing time ($t_{hw} + t_{tr}$). Since t_{hw} is measured based on the latency results from HLS and the frequency, t_{tr} is calculated by subtracting the measurement form the timer IP and t_{hw}.

The t_{hw} and t_{tr} measurements are used as estimation for a framework, in which the transmission link and the FPGA are identical to those used in the Zynq platform, while the timing measurements for the execution time on the ARM core are discarded.

5.2 Speedup on MapReduce

Assuming a platform with M mapper/reducer slots, and n input data splits (n map tasks), each slot runs $\lceil \frac{n}{M} \rceil$ or $\lfloor \frac{n}{M} \rfloor$ map tasks. For simplicity and without loss of generality, we assume that n is a product of M. Thus, each mapper slot runs exactly $\frac{n}{M}$ map jobs. The execution time of the map phase is calculated as follows.

$$T_{map} = \max_{1 \leq m \leq M} \Big(\sum_{i=1}^{\frac{n}{M}} (TF_m^{\frac{n}{M}} - TS_m^1) + \sum_{i=1}^{\frac{n}{M}-1} TI_m^i \Big), \quad (2)$$

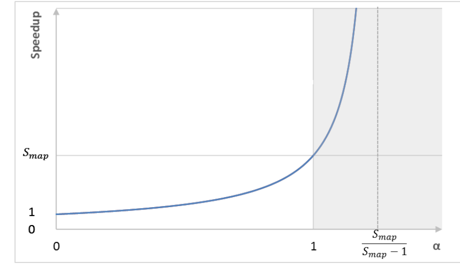


Fig. 4. Overall speedup as a function of α.

where, TS_m^i and TF_m^i are the start and finish times for the m-th slot running the i-th map task, TI_m^i is the time interval for the m-th slot between the end of i-th map task and the start of next map task, and T_{map} is the total time for the map phase. HW+SW acceleration will only speedup the first term in (2).

Lets assume that α is the fraction of time that is accelerated. Then the overall MapReduce speedup is derived from:

$$Speedup = \frac{1}{1 - \alpha(1 - \frac{1}{S_{map}})}, \quad (3)$$

where

$$\alpha = \max_{1 \leq m \leq M} \frac{\sum_{i=1}^{\frac{n}{M}} Tf_m^i - Ts_m^i}{T}, \quad (4)$$

and T is the total execution time.

This methodology was applied to the acceleration of the mapper functions; however, the same procedure is applicable to the accelerations of other computational phases in the MapReduce, including the reduce.

5.3 Upper and Lower Bounds of Speedup

In this section, we analyze the upper and lower bounds for the speedup on an end-to-end Hadoop platform given a function speedup of S_{map}.

Fig. 4 shows the overall speedup as a function of α, which is always lower than 1. Different application types, and system and architecture level configurations yield different values for α. Based on this figure, as α increases, the overall speedup is enhanced. It will approach infinity at $\alpha = \frac{S_{map}}{S_{map}-1}$, which is higher than $\alpha = 1$, and out of the range of acceptable α. The highest acceleration in realized when all the execution time is devoted to the accelerated phase, in which case, an acceleration in the range of S_{map} is obtained.

Considering that not all parts of the map and/or task are best suited for hardware acceleration, and that not all phases of the MapReduce (shuffle, sort, etc.,) are accelerated, the extent of the achievable accuracy is limited. As a result the acceleration can be limited to 20% as in [15], or upto $1.8\times$ in [16], $2.6\times$ in [17], $3\times$ in [18], and $4.1\times$ as shown in Table 3.

6 MapReduce Parallel Implementation in Hadoop Streaming

The machine-learning applications studied in this paper, include various commonly used classification and clustering

2332-7766 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMSCS.2018.2854787, IEEE Transactions on Multi-Scale Computing Systems

algorithms. In this section, we discuss our approach to implement parallel version of these learning algorithms in Hadoop MapReduce streaming.

6.1 Support Vector Machine

SVM is a widely used classification algorithm. In this paper we implement proximal SVM [19]. We assume numerical training data with two classes. Training the classifier is done as follows:

$$\begin{bmatrix}\omega \\ \gamma\end{bmatrix} = (\frac{I}{\nu} + E^T E)^{-1} E^T De, \tag{5}$$

where, I is the identity matrix, e is a vector filled with ones, D is a diagonal matrix of training classes, E is a diagonal matrix with feature vectors, and ν is a scalar constant used to tune the classifier. The classification is done through calculation of $(x^T\omega - \gamma)$, which returns a number, the sign of which corresponds to the class.

Parallelization with MapReduce is achieved through the following property:

$$\begin{aligned} E^T E &= E_0^T E_0 + E_1^T E_1 + \cdots + E_{n-1}^T E_{n-1}, \\ E^T De &= E_0^T D_0 e + D_1^T D_1 e + \cdots + E_{n-1}^T D_{n-1} e, \end{aligned} \tag{6}$$

where the training data is split into n splits. Thus, each mapper calculates one element of the sum in (6), and they all produce the same output key. The reducer calculates the sum and carries out the matrix inversion. Thus, the size of the data splits has a significant influence over the overall execution time as it decides the size of E and D matrices. Fig. 5-a shows the pseudo-code for the map and reduce functions in the MapReduce implementation of the SVM algorithm.

6.2 K-means

K-means is the most commonly used clustering algorithm. It partitions a set of n objects into k clusters to maximize the resulting intra-cluster similarity, while minimizing inter-cluster similarities.

First, the algorithm selects k objects representing initial cluster centers. The remaining objects are assigned to the clusters to which, they are more similar, based on the distance between each object and the cluster center. Subsequently, a new center is calculated for each cluster. This process iterates, with the centers updated in each iteration, until the center values converge. The distance computations between objects and cluster centers comprise the biggest portion of the calculations, which may be done in parallel for different objects. However, the new centers are calculated serially. In MapReduce implementation of the parallel K-means [20], the map function performs the procedure of assigning each sample to the closest center. The reduce function performs the procedure of updating the new centers. Fig. 5-b shows the pseudo-code for the map and reduce functions in the MapReduce implementation of K-means.

6.3 Naive Bayes

Naive Bayes is one of the supervised machine learning classification algorithms, which is based on applying Bayes theorem with the naive assumption of independence between every pair of features. Given a variable y, a feature vector $x_1, x_2, \cdots, x_n$, and the naive independence assumption:

$$\begin{aligned} P(y|x_1,\cdots,x_n) &= \frac{P(y)P(x_1,\cdots,x_n|y)}{P(x_1,\cdots,x_n)} \\ &= \frac{P(y)\prod_{i=1}^{n} P(x_i|y)}{P(x_1,\cdots,x_n)}. \end{aligned} \tag{7}$$

Since the denominator is constant for a given input, the following rule classifies an object with $x_1, \cdots, x_n$ as features.

$$\hat{y} = \arg\max_y \prod_{i=1}^{n} P(x_i|y), \tag{8}$$

where $P(y)$ is the relative frequency of class y in the training set.

Utilizing the maximum a posteriori (MAP) estimation, we set $P(x_i|y)$ to the relative frequency of feature x_i in the training set.

Given a large set of training data in the MapReduce platform, HDFS splits the input data and replicates them to the available nodes in the clusters.

In the MapReduce implementation, the map phase creates a list of <key,value>, where the key is a combination of the class, attribute and the attribute value, namely a unique string and the value is 1. In the reduce task, the values of the same key is added up and a single <key,value> pair is emitted, where the value is the number of occurrences of a specific string in the output of the mappers. Fig. 5-c shows the pseudo-code for the map and reduce functions in the MapReduce implementation of naive Bayes.

6.4 K nearest neighbor

KNN is an algorithm that finds the k nearest neighbors in the training data set for a given point, and classifies it by a majority vote on these k neighbors. The algorithm does not explicitly require a training phase. It involves sorting the data vector coordinates along with the class label. In the testing phase, the aim is to find the class label for the new point. In the MapReduce implementation of KNN [21], the map function calculates the distance of each data point from the training data, and lists out the distances with the corresponding classes. Fig. 5-d shows the pseudo-code for the map and reduce functions in the MapReduce implementation of KNN.

7 HW+SW CO-DESIGN OF THE MAPPER ON ZYNQ

Acceleration of the applications through HW+SW co-design is a complex problem, particularly because different phases of the same application often prefer different configurations and, thus, it requires specific mapping to find the best match. Also the cost of communication and synchronization between the FPGA and the CPU, due to data dependencies, could potentially eliminate the benefit of HW+SW co-design. Therefore, careful mapping between the HW and SW is required to take all these cost overheads into consideration and in fact due to all these overheads not all applications will benefit from HW+SW co-design method.

Fig 6 shows the block diagram of the FPGA+CPU platform used in the implementations. The HW+SW acceleration consists of 3 major parts, namely processing system (PS), interconnects, and programmable logic (PL). The PS

2332-7766 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMSCS.2018.2854787, IEEE Transactions on Multi-Scale Computing Systems

Map
Input: train data containing feature vectors and training classes,
v (a scalar constant to tune the classifier)
Output: <*key*, *value*> pair
0: Create *e* as vector filled with ones;
0: Split the train data into *n* blocks
0: For *i*=1 to *n*:
Create D as a diagonal matrix of training classes;
Create E as a diagonal matrix with feature vectors;
Calculate E^tE;
Calculate E^tDe;
0: Output <*key, value*>;
key: all the mappers produce the same key
value: A string containing the E^tE and E^tDe values

Reduce
Input: *key*, and *V* which is the list of values corresponding to the key
Output: <*key'*, *value'*> pair,
0: $SUM_E^tE=0$;
0: $SUM_E^tDe=0$;
0: For *i*=1 to *n*:
$SUM_E^tE += E^tE$;
$SUM_E^tDe += E^tDe$;
0: Calculate inverse of $(\frac{I}{v} + E^tE)$;
0: Calculate γ and ω based on $\begin{bmatrix}\omega \\ \gamma\end{bmatrix} = (\frac{I}{v} + E^tE)^{-1}E^tDe$;
0:output <*key'*, *value'*>;
key': key
value': string containing values for γ and ω

(a)

Map
Input: Global variable *centers*, the offset *key*, the sample *value*
Output: <*key*, *value*> pair,
key: index of the closest center point
value: string of information
0: Construct the sample *instance* from *value*;
0: *minDis* = *MAX VALUE*, *index* = -1;
0: For i=0 to *#centers:*
dis= *ComputeDist*(*instance*, *centers*[*i*]);
If *dis* < *minDis* :
minDis = *dis;*
index = *I;*
0: Output <*key, value*> pair;
key: index
value: string containing values of all dimensions

Reduce
Input: *key,* which is the index of cluster and *V,* which is the list of
parallel sums (values corresponding to each key)
Output: <*key'*, *value'*> pair,
key': index of the cluster
value': string representing the new centers
0: Initialize the array record with the sum of value of each dimensions
of the samples contained in the sale cluster;
0: *NUM=0;*
0: For *value* in *V*:
NUM+=num;
(add the value of different dimensions of instance to the
array)
0: Divide the entries of the array by NUM;
0: Output <key', value'> pair;
key': key
value': string containing center's coordinates

(b)

Map
Input: train data containing attribute values and class label
Output: <*key*, *value*> pair
0: Load the train data
0: Create a *key* for each train data point, which is a combination of, class,
attribute name and attribute value;
0: Set the value for all the keys to 1;
0: Output <*key, value*>;

Reduce
Input: *key,* and *V* which is the list of values corresponding to each key
Output: <*key'*, *value'*> pair,
0: Sum=0;
0: for each value in *V*:
0: For *i*=1 to *n*:
SUM+=1;
0:output <*key'*, *value'*>;
key': key
value': the number of occurrence of a specific string in the
output of the mappers

(c)

Map
Input: testing data-set and training data-set
Output: <*key*, *value*> pair,
Key: testData
value: string containing information about distance form
trainData with their respective class label
0: *testList=newtestList*;
0: Load *testfile*;
0: *testList* <= *tetfile*; (Update list with data points from file)
0: Open trainfile; (loading train data one at a time)
0: *ComputeDist*(*trainData,testData*);
0*:* write the distance of testData points from all the training data points
with their respective class label.
0: Output <*key, value*> pair;

Reduce
Input: *key,* and *V* which is the list of values corresponding to each key
Output: <*key'*, *value'*> pair,
key': testData point
value': class label
0: load the value of *K*;
0: Load testData points one at a time (load keys and their values);
0: Look through the *K* lowest distances for each testData point and
increment the corresponding class label counter;
0: Assign the class label with the highest count for each testData point;
0: Output <key', value'> pair;
key': key
value': string containing classification information

(d)

Fig. 5. MapReduce Pseudo-code for SVM (a) SVM, (b) kmeans, (c) naive Bayes, and (d) KNN.

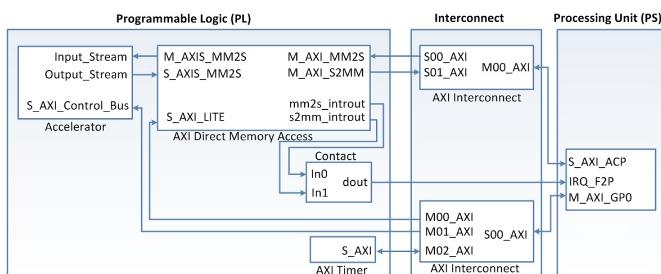


Fig. 6. Block diagram of the FPGA+CPU platform utilized in the acceleration

is the processor, PL is the FPGA and the interconnects are the AXI interconnect IPs, which transfers data between the processor and programmable logic.

The programmable logic consists of 3 major IPs. The accelerator is the main IP generated through Vivado high-level synthesis (HLS). We provide HLS-ready C++ codes, translate them to VHDL and generate the corresponding IP. AXI4-Lite interface is used for sending control data, as AXI-Lite is suitable for sending small amount of data with specific addresses. AXI4-Stream interface is used for sending and receiving input data and result data, since it allows burst transactions as opposed to AXI4-Lite.

AXI Direct memory access (DMA) and AXI timer are

2332-7766 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMSCS.2018.2854787, IEEE Transactions on Multi-Scale Computing Systems

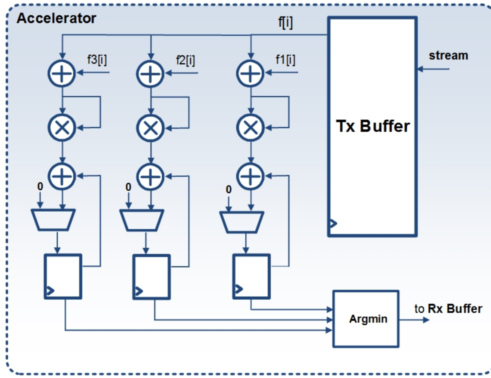


Fig. 7. The accelerator for the K-means mapper.

the other two IPs utilized. The former converts the stream transactions to memory map and allows reading and writing to and from the DDR memory. The latter measures execution time for SW, HW and the transfer time; however, we are only interested in the HW and transfer time, as we use Intel Atom and Xeon cores to execute the SW part.

We perform a full HW+SW implementation for the mapper functions of the studied machine-learning applications. We analyze each application thoroughly by first analyzing the timing of various functions within the map phase of each application using gprof and the perf tool. Subsequently, we selected specific functions for each application to be offloaded to the FPGA.

Fig. 7 shows the details of the FPGA implementation for the K-means mapper with $K = 3$. The features of input objects are transferred from the CPU through the AXI stream, and saved on the Tx buffer. It should be noted, that the transmit and receive streams are stored in a buffer with a data size of 64-bits and depth of 4096 [12]. The features of the incoming objects are read from the buffer, subtracted from the corresponding feature of each cluster center, squared, and accumulated to calculate the Euclidean distance of the object from each cluster center. Subsequently, these value are compared to find the cluster that best matches the incoming object. In Fig. 7, $f[i]$ is the i-th feature value for the incoming object, $fk[i]$ is the i-th feature of k-th cluster center. The *Argmin* block outputs the index of the cluster each object belongs to. Calculation of the Euclidean distance of the objects from all the cluster centers in done in parallel in this figure. Thus, for bigger values of K, a higher level of parallelism can be exploited. Moreover, The following actions can increase the speed of the calculations.

- saving the feature values in 32-bit format, reading two features values in each cycle, and thus parallelizing the calculation of the Euclidean distances.
- pipelining the paths with long delays.
- for big K values, the *Argmin* block can be enhanced by using efficient sorting algorithms.

The same process is used to accelerate mapper functions for other applications, e.g., in the SVM mapper, the calculation of the vector multiplication is carried out on the accelerator. In naive Bayes, the construction of the key string from class, attribute name, and attribute value, and in KNN, the distance calculation of the data points from the training data are moved to the accelerator.

We used Vivado HLS to create FPGA accelerators for these functions. Other IPs including AXI interconnect, DMA were created in the FPGA and connected to the Processing system using Vivado. The FPGA was programmed with the generated bit stream. The map functions were modified to transmit the data to the FPGA through function calls associated with the drivers of the AXI interconnect. AXI specific Control signals are used to notify the completion of the data transfer to and from the FPGA. To maintain the memory coherency we use the accelerator coherence port (ACP). During the ACP writes the cache line is evicted from L1 (if present) and L2 is updated. The DMA transfer writes to L2 so does not pollute the L1 cache. Assuming data is in the processor cache, the access is low-latency using ACP. However, to use the High-performance port (HP), memory coherency can be established by flushing the cache from the PS for each transmission. It should be noted that memory transfer time is a key determining factor for performance and energy efficiency. However, the main focus in this work is the exploration of the block of core, cache, and FPGA architecture. DRAM and LLC are out of scope the paper due to the limitation of measuring the breakdown of data transfer time from DRAM to shared LLC and from LLC to Core/FPGA.

Table 2 shows the FPGA resource utilization, (which includes the accelerator IP, DMA, timer and the interconnect) and the maximum frequency of accelerators. It is worth mentioning that the reported frequencies reflect the processing power of Artrix-7 and higher frequencies can be achieved using high-end FPGAs.

Moreover, Table 2 shows the acceleration speedup of the map function after HW+SW co-design with respect to their software implementation (measured by the perf tool on Intel Atom and Intel Xeon) based on equation (1).

Speedup values were calculated by measuring t_{sw} for Atom and Xeon, t_{hw} and t_{tr} as described in Section 5.1.

It should be noted that for each application, the computation intensive part of the applications is accelerated for only one map task. Based on Table 2 not all applications use all the FPGA resources. For these applications (and other applications, assuming larger FPGAs) due to the highly parallel and data-independent nature of MapReduce multiple map tasks can be executed in parallel; however the bandwidth of the transmission link between the FPGA and CPU will eventually become the bottleneck.

Table 2 shows that not all applications show high potential for acceleration on the this platform. This is due to dependencies and communication as well as synchronization cost in different parts of the code, and the fact that some applications have a smaller computational part, resulting in a lower speedup. For SVM for instance, while the computation part of the map function is accelerated by offloading to the FPGA, still most of the parts remain on the core, allowing negligible change in the overall execution time after the hardware acceleration. For the K-means, KNN and Naive Bayes applications on the other hand, the range of speedup is higher. Thus, applications with *higher numerical calculations* and *higher levels of parallelism* are better candidates for being offloaded on the FPGA.

8 EXPERIMENTAL SETUP

A parallel MapReduce version of the 4 studied applications

2332-7766 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMSCS.2018.2854787, IEEE Transactions on Multi-Scale Computing Systems

TABLE 2
Resource utilization and timing of HW+SW acceleration IPs for map functions.

Resources	Available	Utilization (%)			
		SVM	K-means	KNN	Naive Bayes
FF	35200	7.57	6.38	4.07	4.82
LUT	17600	12.03	11.98	7.13	8.66
Memory LUT	6000	1.23	2.03	1.14	1.23
BRAM	60	59.29	37.86	47.50	62.14
DSP48	80	2.27	9.55	0.00	0.00
BUFG	32	3.13	3.13	3.13	3.13
Frequency [MHz]		321	333	342	301
throughput [MBps]		25.3	59.3	36.6	95.3
$T_{hw}[\mu s]$		14	15	156	21
$T_{tr}[\mu s]$		103	36	707	59
$T_{sw}[\mu s]$	Xeon	87	31	424	63
	Atom	122	42	1027	147
$T_{map}[\mu s]$	Xeon	210	346	3230	1284
	Atom	249	737	7371	4202
S_{map}	Xeon	1.028	4.20	2.51	8.99
	Atom	1.042	7.90	3.90	18.52

TABLE 3
Changing the number of mapper slots

cores	mappers	total_time(s)	map_time(%)	accelerated_time(s)	speedup
Naive Bayes					
x	1	756.59	85.53	182.68	4.142
x	4	349.78	49.82	195.42	1.790
x	8	290.46	31.28	209.87	1.384
x	12	301.12	38.46	198.36	1.518
a	1	1,256.32	82.55	275.48	4.561
a	4	482.15	56.74	223.85	2.154
a	8	495.05	44.39	287.25	1.723
K-means					
x	1	194.87	89.43	62.64	3.111
x	4	62.75	71.90	28.62	2.192
x	8	41.34	58.24	23.19	1.783
x	12	47.59	56.85	27.18	1.751
a	1	393.31	94.50	68.93	5.706
a	4	108.66	88.54	24.74	4.393
a	8	60.22	79.18	18.60	3.238
KNN					
x	1	1,737.53	93.36	764.55	2.273
x	4	581.18	74.50	321.43	1.808
x	8	390.70	56.09	259.32	1.507
x	12	369.87	58.33	240.45	1.538
a	1	3,921.08	94.74	1158.88	3.384
a	4	1,145.62	86.50	408.79	2.802
a	8	707.14	71.27	332.66	2.126
SVM					
x	1	120.81	87.03	118.76	1.017
x	4	46.20	65.10	45.61	1.013
x	8	35.19	42.73	34.89	1.008
x	12	101.74	79.72	100.15	1.016
a	1	267.99	93.85	258.32	1.037
a	4	83.17	83.60	80.49	1.033
a	8	51.90	73.26	50.44	1.029

was implemented using Hadoop Streaming and profiled for 1GB using Intel VTune [22] on Atom C2758 and Intel Xeon E5-2440. The map functions of 4 machine learning and data mining applications were accelerated and the speedup rate for the map functions were calculated for the Xeon and Atom architectures assuming integration of Artrix-7 FPGAs through on-chip AXI-interconnect streaming as described in Sec 7. In order to calculate the potential speedup on the Hadoop platform after the acceleration of map functions, the speedup of the overall MapReduce is calculated based on equation (4), assuming the map function is accelerated with the calculated rates in Sec 7.

9 Acceleration Results

9.1 Execution time and speedup

The execution time, power, and energy efficiency of an application is a factor of several parameters at the system, architecture and application levels. For selection of an optimum design, the sensitivity of the design's performance and power to various parameters is of high importance. In this section, we analyze various parameters including the number of mapper slots and the size of input data. All the experiments are performed and the results were collected for both Xeon as well as Atom architecture.

9.1.1 Number of Mapper/Reducer Slots

One of the important criteria while making architectural decisions is the restrictions on the number of mapper/reducer slots. The MapReduce programmer decides the number of map and reduce tasks. First, the input data is split into data splits. Then, based on the data split size, the number of map tasks is derived. However, The number of tasks that are executed in parallel depends on the hardware resources, namely mapper/reducer slots.

Different techniques are used to determine the optimal number of slots based on the architecture. Mostly, experiments show that for performance optimization the number of slots is best to be tuned in a range of $(0.95 - 1.75)\times$ the number of available cores [23]. In this case, all the cores in the architecture are busy.

Table 3 shows the results for execution time and speedup on Atom and Xeon with different number of mapper/reducer slots. In Table 3, *map−time(%)* shows the execution time of the map phase with respect to the total execution time (i.e., α), the HDFS block size is 64MB, and the input data size is 1GB. The number of reduce tasks is set to one. Thus, only one slot is taken up by the reduce task after the completion of map, shuffle and reduce phases. It should be noted that Xeon and Atom have 12 (dual-socket) and 8 cores, respectively. Thus, the experiments were carried out for 1, 4, 8 and 12 slots on Xeon, and 1, 4 and 8 slots on Atom.

Table 3 shows that mostly, the execution time before the acceleration decreases with the increasing number of slots, which is due to the higher exploitation of the parallelism inherent in the MapReduce framework.

Table 3 shows that the speedup realized on both machines drops noticeably with the increasing number of mapper slots. This is to be expected, as the fraction of time spent in the map phase i.e., α, is reduced with increasing the number of mappers. Also more time is spent in other phases like shuffle as well as transferring of data, as the number of mappers increases.

Since the final execution time is both a function of the HW+SW speedup gain of the map function i.e., S_{map}, as well as α, the lowest post-acceleration execution time is case-specific, and is decided not only by the nature of the application and the potential HW acceleration in the code, but also by the system and architecture level parameters such as number of slots and the choice of CPU core. For instance with the SVM and K-means, on both Xeon and Atom the optimal number of mapper/reducer slots is 8, while the optimal configurations are different for Naive Bayes and KNN.

It should be noted that Intel Xeon has two sockets of 6 cores. By using more than 6 cores (mappers) the increasing communication time between the two sockets results in the a number of counter-intuitive behavior in which increasing the number of mappers beyond 6 slightly increases

2332-7766 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMSCS.2018.2854787, IEEE Transactions on Multi-Scale Computing Systems

the execution time for some applications. Interestingly, the performance gap between configurations with different number of mapper/reducer slots reduces significantly after acceleration. For example in naive bayes and on Atom, the execution time before acceleration is almost 250% different on 1 mapper compared to 4 mappers. However this gap drops to less than only 20% after acceleration. In some cases after acceleration, the trend even changes; For instance, in naive Bayes and on Xeon core, while the configuration with 1 mapper/reducer slot is significantly slower than 4, it becomes faster after the acceleration. This is in part, due to the fact that the map phase in the configuration with 1 slot accounts for significantly larger part of the total execution time compared to the configuration with four slots. This observation is very important and can be leveraged to guide the co-scheduling decision of multiple applications, i.e. while scheduling multiple applications competing for mapper/reducer slots, HW+SW acceleration reduces the reliance on large number of slots (availabe cores) for performance gain. In fact, HW+SW acceleration allows a configuration with fewer number of slots to be competitive with the one with larger number of slots. As a result, more cores are freed up on each node to accommodate the scheduling of incoming applications in a cloud-computing environment.

Another observation from Table 3 is the reduction in the performance gap between Atom and Xeon. The aggressive superscalar architecture of Xeon allows it to process jobs at a higher speed compared to Atom. However, the acceleration yields lower speedup gains on Xeon. In few configurations this results in Atom having a comparable speed to Xeon after acceleration. For instance, with 8 mapper/reducer slots, the K-means application is initially slower on Atom, however after the acceleration, its execution time is comparable to Xeon. This is due to the fact, that the map phase accounts for 58% and 88% of the total execution time on Xeon and Atom, respectively. As a result a larger portion of the application is accelerated on Atom at a higher rate ($7.9\times$ vs. $4.2\times$), making its performance comparable to Xeon. This is an important observation for making architectural decisions for the choice of CPU in presence of hardware accelerator.

9.2 Power and Energy efficiency

An important benefit of HW+SW acceleration is the improvement in the energy-efficiency. General-purpose CPUs such as Atom and Xeon are not designed to provide maximum efficiency for every application. Accelerators help improve the efficiency by not only speeding up execution time, but also executing the task with just enough required hardware resources. To this end, we measure the power and calculate the energy delay product (EDP) both before and after the acceleration.

The overall power values were calculated with the same methodology as the one used to calculate the accelerated execution time. Wattsup pro power meter [24] was used for power readings on Xeon and Atom servers. We measured the average power for individual phases on Xeon and Atom using Wattsup pro power meter. Moreover, for each mapper function on the FPGA board, we used picoScope digital oscilloscope. We multiplied the average power by the execution times before/after the acceleration to get the corresponding energy values. These number were used to calculate the energy consumption. Subsequently, the energy consumption of the accelerated mappers was replaced with that of the FPGA to estimate the energy of the accelerated design. Power measurement for the hardware part of each mapper function was performed using picoScope digital oscilloscope for the FPGA board. We measured the power by measuring the current flowing to FPGA and multiplying that by the voltage. To measure the current, we measured the voltage drop across the test points provided on the FPGA board divided by the resistance around those points. By averaging the resulting energy consumption over the new execution time, the new power values were calculated.

TABLE 4
Power end energy efficiency results

cores	mappers	power(w)	power,acc(w)	power ratio	EDP(kws^2)	EDP,acc(kws^2)	EDP ratio
Naive Bayes							
x	1	27.34	21.19	1.29	1565.02	707.15	2.21
x	4	36.49	34.48	1.06	4464.12	1316.62	3.39
x	8	38.37	37.61	1.02	3237.16	1612	2.00
a	1	3.56	6.71	0.75	5622.88	994.54	5.65
a	4	5.55	7.28	0.76	1289.79	364.63	3.54
a	8	5.94	7.89	0.53	1475.30	458.56	6.58
K-means							
x	1	12.99	14.43	0.91	493.33	55.92	8.82
x	4	18.15	20.19	0.90	71.47	16.54	4.32
x	8	37.11	41.22	0.91	63.42	5.78	2.88
a	1	3.65	7.45	0.49	565.17	35.39	15.97
a	4	5.74	11.55	0.50	67.74	67.74	9.58
a	8	4.27	12.20	0.35	15.48	4.21	3.67
KNN							
x	1	27.69	20.66	1.34	83602.92	12043.49	6.94
x	4	44.55	37.50	1.19	15047.37	3874.45	3.88
x	8	50.22	46.07	1.09	7667.04	3098.67	2.48
a	1	3.53	4.90	0.72	54273.28	6580.51	8.23
a	4	5.49	8.03	0.68	7205.15	1341.17	5.37
a	8	7.57	11.30	0.67	3785.35	1251.15	3.02
SVM							
x	1	21.89	20.51	1.07	319.52	289.19	1.10
x	4	26.05	24.81	1.05	55.59	51.62	1.08
x	8	24.62	23.88	1.03	30.48	29.06	1.05
a	1	3.42	3.38	1.01	245.79	225.81	1.09
a	4	4.50	4.41	1.02	31.15	28.57	1.09
a	8	3.56	3.52	1.01	9.58	8.96	1.07

Table 4 shows the results for average power and EDP, where *power* and *power,acc* account for power before and after the acceleration, respectively, and *EDP* and *EDP,acc* represent EDP before and after the acceleration, respectively. *Power ratio* and *EDP ratio* show the ratio of the power and EDP before the acceleration to the power and EDP after the acceleration, respectively.

Results show that the power mostly decreases for Xeon, since a part of the task of high-power Xeon cores is moved to the low power FPGA. This does not apply to Atom, in which the power consumption increases after the acceleration, as the power consumption of FPGA board is comparable to the power of low power Atom core. However, EDP which is an indicator of the energy efficiency is reduced significantly (in some configurations by upto $16\times$).

As discussed earlier, the speedups on the Atom architecture are higher compared to Xeon; As a result, the improvement in the energy-efficiency is more significant on Atom. Moreover, as the number of mapper/reducer slots increases, i.e., the CPU utilization increases on both Atom and Xeon, the energy-efficiency gain of acceleration reduces. This is somewhat expected, as increasing the number of mapper/reducer slots results in an increase in the number of active cores on each machine which in turn, increases the utilization of the CPU as well as its energy-efficiency, therefore bringing smaller energy-efficiency gain after the acceleration. In addition, as discussed earlier, increasing the number of mapper/reducer slots significantly reduces the performance gain after acceleration compared to be-

2332-7766 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMSCS.2018.2854787, IEEE Transactions on Multi-Scale Computing Systems

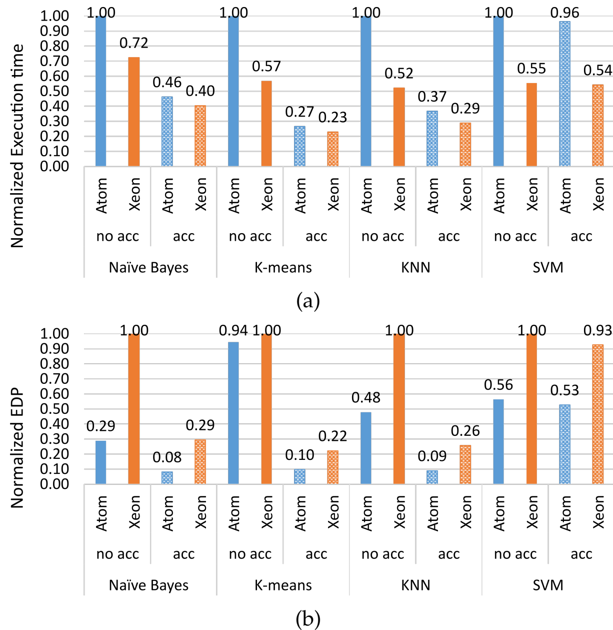


Fig. 8. Normalized [a] Execution time [b] EDP comparison of big and little core.

TABLE 5
Execution time for various scheduling schemes.

Workload	W1		W2		W3		W4	
	Xeon	Atom	Xeon	Atom	Xeon	Atom	Xeon	Atom
Fair Serial [s]	1064	1763	1668	2853	1924	3296	2273	3943
Fari Parallel [s]	739	1311	1149	2089	1301	2219	1561	2758
Parameter-tuned [s]	720	1295	1125	2070	1269	2201	1549	2740
Fair Serial, SW+HW [s]	760	965	1171	1516	1346	1753	1582	2067
Fair Parallel, SW+HW [s]	447	739	665	843	741	917	889	1109
Parameter-tuned, SW+HW [s]	303	409	486	684	491	712	789	1093

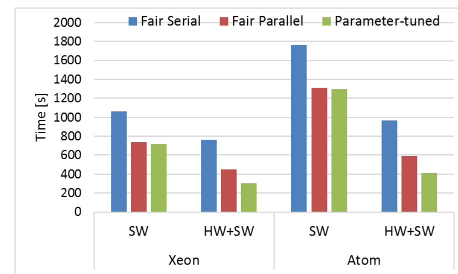


Fig. 9. Execution time for various scheduling schemes for $W1$ workload.

fore acceleration. Therefore, for some cases, increasing the number of mapper/reducer slots yields small changes in the performance with a significant increase in the power. For such cases, the lower number of mapper/reducer slots after acceleration is the best configuration to maximize the energy-efficiency.

9.3 Big and Little Core in Presence of acceleration

The simulation results suggests that the hardware acceleration reduces the gap between Atom and Xeon. In this section we compare the execution time and EDP of the studied benchmarks for Atom and Xeon. Fig. 8 compares the execution time and EDP. All the execution times have been normalized to the execution time on Xeon and all the EDPs have been normalized to Xeon. As expected the execution time and EDP are lower on Atom. However the figures shows that both the execution time gap and the EDP gap between Atom and Xeon is significantly reduced. This shows that the hardware acceleration targets the speed of the little cores more significantly, while its impact on energy-efficiency is more significant on big cores.

10 Scheduling in Presence of Hardware Accelerator

In this section, we show how hardware acceleration changes scheduling decision and whether it provides more opportunity for fine-tuning of optimization parameters to further enhance performance, when co-scheduling multiple MapReduce applications. In order to study the impact of parameter-aware hardware acceleration on scheduling, we create a workload consisting of the studied applications. We create several workloads $W(M,N,O,P)$, with $M\times$KNN, $N\times$Kmeans, $O\times$NB, and $P\times$SVM. For simplicity we assume there are a total of 12 benchmarks to be scheduled in each workload. We randomly picked up the following workloads to present how much opportunity exist for optimized scheduling. The following workloads were studied: $W1(1,5,1,5), W2(2,4,4,2), W3(2,4,3,3)$ and $W4(3,3,3,3)$. We carry out our simulations on a 8-core machine, (thus with 8 mapper/reducer slots) and use fair serial, fair parallel and parameter-aware scheduling methods for both accelerated and pure software MapReduce implementations. In the fair serial scheduling, we allow each application to use all the 8 cores and schedule them serially. In the fair parallel scheduling, we allow each application to use 4-cores, thus 2 applications can run in parallel. In the parameter-aware scheduling, we minimize the time of the whole workload by tuning the parameters (number of cores and HDFS block size).

Table 5 shows the results for the studied workloads. The results show that tuning the parameters enhances the performance of hardware-accelerated framework even further. In order to make a comparative analysis between pure software and post-acceleration, the results for W1 are depicted in Fig. 9. Based on the results, for both Atom and Xeon the execution time of fair parallel scheduling is less, both before and after hardware acceleration. The hardware acceleration enhances the speed of the workload on both machines. Moreover the parameter-tuned scheme yields the best results. In this scheme, not only the parameters are tuned to get the lowest execution time, the resources are distributed not fairly, but based on the benchmark performance. Specifically for some benchmarks, the execution time after the acceleration is significantly reduced when they are consuming less hardware (e.g., when they are running only on one core). This allows us to run these benchmarks on one core and leave other cores free for other programs. For instance, for the studied workload, KNN takes up 4 cores, and Naive Bayes, K-means and SVM occupy three cores, leaving one core for other applications.

More importantly, Fig. 9 shows, that while the effect of fine-tuning is negligible before the acceleration (i.e., an average of 2% and 0.9% improvement on Xeon and Atom, respectively over fair parallel scheduling for the 4 studied workloads); its impact on the post-acceleration performance is significant (i.e., an average of 37% and 24% improve-

2332-7766 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMSCS.2018.2854787, IEEE Transactions on Multi-Scale Computing Systems

TABLE 6
Results for various input data sizes on a 12-node cluster.

Input(GB)	mappers	total_time(s)	map_time(%)	accelerated_time(s)	speedup
			Naive Bayes		
1	8	93	63.44	40.56	2.293
5	40	401	52.62	213.47	1.878
10	71	583	41.17	369.70	1.577
20	69	1256	46.02	742.29	1.692
100	71	6031	47.55	3482.65	1.732
			K-means		
1	8	20	85.00	7.05	2.838
5	40	49	93.88	13.95	3.512
10	78	62	95.16	17.05	3.637
20	78	87	95.40	23.76	3.661
100	85	383	97.65	98.05	3.906
			KNN		
1	8	245	87.35	116.26	2.107
5	40	577	78.86	303.27	1.903
10	69	872	41.17	438.62	1.329
20	80	1297	74.50	715.86	1.812
100	82	4849	78.67	2553.90	1.899
			SVM		
1	8	18	83.33	17.59	1.023
5	40	21	71.43	20.59	1.020
10	71	37	81.08	36.18	1.023
20	76	51	70.59	50.02	1.020
100	77	173	88.44	168.83	1.025

ment on Xeon and Atom, respectively over fair parallel scheduling). Thus, hardware acceleration provides more opportunities for fine-tuning of optimization parameters to further enhance the performance.

11 Scalability in a Multi-node Architecture

In order to understand the scalability of the acceleration method presented in this paper, in this section, we study HW+SW acceleration in a multi-node cluster and across a large range of input data size. We use a 12-node (1 NameNode and 11 DataNodes) cluster, each with dual Intel Xeon E5-2670 (2.60GHz) 8 core CPUs allowing upto 176 mapper/reducer slots. We execute the applications for various data sizes. The HDFS block size is set to 128MB. Thus, based on the input data size, the number of data splits vary (i.e, 8, 40, 80, 160 and 800 for input data of 1, 5,10, 20 and 100GB, respectively). Table 6 shows the result collected on the cluster, where *mapper* shows the number of occupied slots among the available slots. The rest of the slots are utilized for other tasks, including reduce. Table 6 shows that the HW+SW acceleration of mapper functions, in a cluster is as effective as their acceleration of a single-node platform.

Also as shown in Table 6, the execution times before and after the acceleration changes semi-logarithmically with the size of input data. Also note that while in some cases the overall speed up after acceleration increases as the size of data increases (e.g., in K-means), in other cases the speed up reduces as the size of data grows. In fact changing the size of data changes the amount of intra-node communication and consequently affects α, which is the contribution of map time to the total time.

12 Related Work

The performance and bottlenecks of Hadoop MapReduce have been extensively studied in recent work [6], [25]–[29]. To enhance the performance of MapReduce and based on the bottlenecks found for various applications, hardware accelerators are finding their ways in system architectures.

MARS [30], a MapReduce framework on an NVIDIA G80 GPU was shown to be up to 16 times faster than a quad-core CPU-based implementation for six common web applications. In [31], GPU MapReduce (GPMR) is presented as a stand-alone MapReduce library that modifies the typical MapReduce tasks to fit into GPMR. While the GPU-based platforms have achieved significant speedup across a wide range of benchmarks, their high power demands preclude them for energy-efficient computing [32]. Alternatively, FPGAs have shown to be more energy efficient [33]. Moreover, they allow the exploitation of fine-grained parallelism in the algorithms. It should be noted that current deep Neural Networks rely heavily on dense floating-point matrix multiplication, which maps well to GPUs. While current FPGAs offer superior energy efficiency, but they do not offer the performance of todays GPUs on DNNs. However, advances in the FPGA technology, suggest that new generation of FPGAs will outperform GPUs. In [34] the authors compare two generations of Intel FPGAs (Arria 10, Stratix10) against the latest highest performance Titan X Pascal GPU. For a ResNet case study, their results show that for Ternary ResNet [35], the Stratix 10 FPGA can deliver 60% better performance over Titan X Pascal GPU, while being 2.3× better in performance/watt showing that FPGAs may become the platform of choice for accelerating next-generation DNNs.

Various platforms have been deployed to leverage the power of FPGA for acceleration and energy-efficiency purposes [36]–[38]. Traditionally, the integration of the FPGA to CPU has been realized through the PCI-e. In Microsoft Catapult project [37] a composable, reconfigurable fabric was built to accelerate portions of large-scale software services, which consists of 6 × 8 2-D torus of Stratix V FPGAs embedded into a half-rack of 48 machines. The FPGAs are accessible through PCIe. The catapault platfrom is further improved in the Configurable Cloud [39] to allow the datapath of cloud communication to be accelerated with programmable hardware too. Another example is the Alpha Data FPGA board with Xilinx FPGA fabric. In Alpha Data, accelerators are developed in C/C++ and OpenCL languages through the Xilinx SDAccel development environment [40].

Alternatively, hybrid chips that integrate FPGAs with processors reduce the overhead of data transfers, allowing low-cost on-chip communication between the two. Heterogeneous architecture research platform (HARP), is one such platform that integrates Intel CPU with Altera FPGAs [41] through QPI. Another example is Zynq-7000 SoC platform, which integrates ARM cores with Xilinx FPGAs through AXI-interconnect.

In [36], a MapReduce framework on FPGA (FPMR) is described in which, hardware accelerators are introduced for RankBoost application, along with an on-chip processor scheduler that maximizes the utilization of computation resources. In [42], a MapReduce framework on FPGA accelerated commodity hardware is presented, which consists of FPGA-based worker nodes operating extended MapReduce tasks to speed up the computation process, and CPU-based worker nodes, which run the major communications with other worker nodes. In [43], a MapReduce programming model is evaluated that exploits the computing capacity in a cluster of nodes equipped with hardware accelerators (i.e.,

2332-7766 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMSCS.2018.2854787, IEEE Transactions on Multi-Scale Computing Systems

cluster of Cell BE processors).

In [6], a hardware accelerated MapReduce implementation of Terasort is proposed on Tilera's many core processor board. In this architecture, data mapping, data merging and data reducing are offloaded to the accelerators. In Zcluster [15], hardware acceleration of FIR is explored through an eight-salve Zynq-based MapReduce architecture. In [16], a configurable hardware accelerator is used to speed up the processing of reduce tasks in MapReduce framework. They showed upto $1.8\times$ system speedup of the MapReduce applications. In [44], a detailed MapReduce implementation of the K-means application is presented.

In [18] the authors provide programming and runtime support for enabling easy and efficient deployments of FPGA accelerators in data centers and improve the system throughput by upto $3\times$. In [17] the authors explore an FPGA-enabled Spark cluster that features batch processing to alleviate communication overhead. The share FPGAs among multiple CPU threads and improve the performance of DNA sequencing applications by $2.6\times$ compare to a CPU-only cluster.

In [45], microarchitectural characteristics of state-of-the art PCIe-based ([37], [39], [46]) and QPI-based ([41], [47]) platforms has been evaluated. They show that the on-chip integrated QPI-based platform expresses impressive advantage on fine grained communication latency (<4KB). It should be noted that, that the data-access pattern of map functions is highly regular and data in-dependent. This allows offloading of small sub-functions to the FPGA while each sub-function requires a low-volume data transfer and thus low communication latency. Thus this paper, we evaluate how offloading compute-intensive sub-functions within map functions to on-chip integrated FPGA accelerates the MapReduce Applications. In addition, we analyzes how various parameters affects the benefits of HW acceleration.

It should be noted that Hadoop and Spark are two major open source projects for handling big data analytics. Spark can do it in-memory, while Hadoop MapReduce has to read from and write to a disk. Thus, Spark may be up to 100 times faster. On the other hand, Hadoop MapReduce is able to work with larger and distributed data sets than Spark [48]. In this study, we focus on improving the performance of the applications on the MapReduce. However, the same methodology can be applied to computation-intensive phases of Spark applications.

13 Conclusions

In this paper we demonstrate the performance and energy-efficiency advantages of FPGA acceleration for Hadoop applications to find architectural insight and understand the implications of hardware acceleration on various architectural trade-offs in a heterogeneous CPU+FPGA architecture. We evaluated the Hadoop MapReduce on a 12-node server equipped with FPGA hardware accelerators. We offload the mapper to FPGA and fully implement the hardware accelerated functions on the FPGA board. We measured power, and execution time on the server and the FPGA board. We also account for the interconnection overhead between FPGA and the CPU core. We accelerated SVM, K-means, KNN and naive Bayes in this framework. The results show promising speedups as well as energy-efficiency gains of upto $5.7\times$ and $16\times$, respectively using a semi-automated high level synthesis method scalable for cloud computing infrastructure. We further studied how application, system, and architecture level parameters affect the performance and power-efficiency benefits of Hadoop Streaming hardware acceleration. The results show that HW+SW acceleration yields higher speedup on little Atom cores, therefore significantly reducing the performance gap between little and big cores after acceleration. This is due to the fact that on Atom the map phase accounts for a higher portion of the execution time. As a result a larger portion of the application is accelerated on Atom at a higher rate, making its performance comparable to Xeon. The results show that hardware accelerator solution significantly improves the energy-efficiency on high performance core, and substantially enhances the performance on low power cores, therefore simultaneously bridging the performance and energy-efficiency gap between the two architectures. In addition, in presence of hardware acceleration we can reduce the number of mapper/reducer slots (active cores) and yet be as energy-efficient as a case in which, all the available cores are active without any performance loss. This is most beneficial for scheduling decisions by significantly freeing up cores on each node to accommodate scheduling of other incoming applications in a cloud-computing environment.

References

[1] K. Neshatpour, M. Malik, and H. Homayoun, "Accelerating machine learning kernel in hadoop using fpgas," in *2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, May 2015, pp. 1151–1154.

[2] K. Neshatpour, A. Sasan, and H. Homayoun, "Big data analytics on heterogeneous accelerator architectures," in *2016 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, Oct 2016, pp. 1–3.

[3] Ferdman and et al., "Clearing the clouds: A study of emerging scale-out workloads on modern hardware," *SIGPLAN Not.*, vol. 47, no. 4, pp. 37–48, Mar. 2012.

[4] Gutierrez and et al., "Integrated 3d-stacked server designs for increasing physical density of key-value stores," in *ASPLOS*. ACM, 2014, pp. 485–498.

[5] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," in *Proc. conf symp operation systems design and implementation*, 2004.

[6] T. Honjo and K. Oikawa, "Hardware acceleration of hadoop mapreduce," in *2013 IEEE Int. Conf. Big Data*, Oct 2013, pp. 118–124.

[7] "Accelerating hadoop applications using intel quick-assist technology," http://www.intel.com/content/dam/www/public/us/en/documents/solution-briefs/accelerating-hadoop-applications-brief.pdf, accessed: 2014-11-30.

[8] Hardavellas and N. et.al, "Toward dark silicon in servers," *IEEE Micro*, vol. 31, pp. 6–15, 2011.

[9] "Running c++ programs on hadoop," http://cs.smith.edu/dftwiki/index.php/Hadoop_Tutorial_2.2_--_Running_C%2B%2B_Programs_on_Hadoop.

[10] "Hadoop programming with arbitrary languages," https://rcc.fsu.edu/docs/hadoop-programming-arbitrary-languages.

[11] T. White, *Hadoop: The Definitive Guide*, 1st ed. O'Reilly Media, Inc., 2009.

[12] "Logicore axi dma v7.1," *Product Guide for Vivado Design Suite*, Dec 2013.

[13] L. A. Cortes, L. Alej, R. Corts, P. Eles, and Z. Peng, "A survey on hardware/software codesign representation models," 1999.

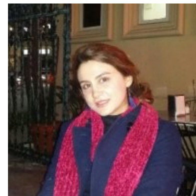
[14] P. Eles and et. al, "Scheduling of conditional process graphs for the synthesis of embedded systems," in *Proceedings of the Conference on Design, Automation and Test in Europe*, ser. DATE '98. Washington, DC, USA: IEEE Computer Society, 1998, pp. 132–139. [Online]. Available: http://dl.acm.org/citation.cfm?id=368058.368119

[15] Z. Lin and P. Chow, "Zcluster: A zynq-based hadoop cluster," in *Int. Conf. Field-Programmable Technology (FPT)*, Dec 2013, pp. 450–453.

2332-7766 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMSCS.2018.2854787, IEEE Transactions on Multi-Scale Computing Systems

[16] C. Kachris and et al., "A configurable mapreduce accelerator for multi-core fpgas," in *FPGAs*, 2014, pp. 241–241.
[17] Y. Liu, J. Yang, Y. Huang, L. Xu, S. Li, and M. Qi, "Mapreduce based parallel neural networks in enabling large scale machine learning," *Computational intelligence and neuroscience*, vol. 2015, p. 1, 2015.
[18] M. Huang, D. Wu, C. H. Yu, Z. Fang, M. Interlandi, T. Condie, and J. Cong, "Programming and runtime support to blaze fpga accelerator deployment at datacenter scale," in *Proceedings of the Seventh ACM Symposium on Cloud Computing*. ACM, 2016, pp. 456–469.
[19] G. Fung and O. L. Mangasarian, "Incremental support vector machine classification," in *ACM SIGKDD*, 2001, pp. 77–86.
[20] Zhao and et al., "Parallel k-means clustering based on mapreduce," in *CLOUD*, 2009, pp. 674–679.
[21] Prajesh and et al., "The k-nearest neighbor algorithm using mapreduce paradigm," in *ICSMO*, 2014, pp. 513–518.
[22] "Intel vtune amplifier xe performance profiler." http://software.intel.com/en-us/articles/intel-vtune-amplifier-xe/, accessed: 2014-11-30.
[23] J. Lin and C. Dyer, *Data-intensive Text Processing with MapReduce*. Morgan and Claypool Publishers, 2010.
[24] "Wattsuppro power meter," http://www.wattsupmeters.com/secure/index, accessed: 2014-11-30.
[25] D. Jiang and et al., "The performance of mapreduce: An in-depth study," *Proc. VLDB Endow.*, vol. 3, no. 1-2, pp. 472–483, Sep. 2010.
[26] C. Tian and et al., "A dynamic mapreduce scheduler for heterogeneous workloads," in *GCC*, Aug 2009, pp. 218–224.
[27] M. Zaharia and et al., "Improving mapreduce performance in heterogeneous environments," in *OSDI*, 2008, pp. 29–42.
[28] K. Neshatpour, M. Malik, M. A. Ghodrat, A. Sasan, and H. Homayoun, "Energy-efficient acceleration of big data analytics applications using fpgas," in *2015 IEEE International Conference on Big Data (Big Data)*, 2015, pp. 115–123.
[29] M. Malik, K. Neshatpour, T. Mohsenin, A. Sasan, and H. Homayoun, "Big vs little core for energy-efficient hadoop computing," in *2017 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2017, pp. 1480–1485.
[30] B. He and et al., "Mars: A mapreduce framework on graphics processors," in *PACT*, 2008, pp. 260–269.
[31] J. A. Stuart and et al., "Multi-gpu mapreduce on gpu clusters," in *IPDPS*, Washington, DC, USA, 2011, pp. 1068–1079.
[32] L. Stolz and et al., "Energy consumption of graphic processing units with respect to automotive use-cases," in *ICEAC*, Dec 2010, pp. 1–4.
[33] J. Fowers and et al., "A performance and energy comparison of fpgas, gpus, and multicores for sliding-window applications," in *FPGA*, 2012, pp. 47–56.
[34] E. Nurvitadhi, G. Venkatesh, J. Sim, D. Marr, R. Huang, J. O. G. Hock, Y. T. Liew, K. Srivatsan, D. Moss, S. Subhaschandra *et al.*, "Can fpgas beat gpus in accelerating next-generation deep neural networks?" in *FPGA*, 2017, pp. 5–14.
[35] A. Kundu, K. Banerjee, N. Mellempudi, D. Mudigere, D. Das, B. Kaul, and P. Dubey, "Ternary residual networks," *arXiv preprint arXiv:1707.04679*, 2017 .
[36] Y. Shan and et al., "FPMR: Mapreduce framework on FPGA," in *FPGA*, 2010, pp. 93–102.
[37] A. Putnam and et al., "A reconfigurable fabric for accelerating large-scale datacenter services," in *ISCA*, June 2014, pp. 13–24.
[38] K. Neshatpour, M. Malik, M. A. Ghodrat, and H. Homayoun, "Accelerating big data analytics using fpgas," in *Field-Programmable Custom Computing Machines (FCCM), 2015 IEEE 23rd Annual International Symposium on*. IEEE, 2015, pp. 164–164.
[39] A. M. Caulfield, E. S. Chung, A. Putnam, H. Angepat, J. Fowers, M. Haselman, S. Heil, M. Humphrey, P. Kaur, J.-Y. Kim *et al.*, "A cloud-scale acceleration architecture," in *Microarchitecture (MICRO), 2016 49th Annual IEEE/ACM International Symposium on*. IEEE, 2016, pp. 1–13.
[40] "Sdaccel development environment [online]," http://www.xilinx.com/products/design-tools/software-zone/sdaccel.html.
[41] Intel, "Accelerator abstraction layer software programmer?s guide."
[42] D. Yin and et al., "Scalable mapreduce framework on fpga accelerated commodity hardware," in *Internet of Things, Smart Spaces, and Next Generation Networking*, 2012, vol. 7469, pp. 280–294.
[43] Y. Becerra and et al., "Speeding up distributed mapreduce applications using hardware accelerators," in *ICPP*, Sept 2009, pp. 42–49.
[44] Y.-M. Choi and H.-H. So, "Map-reduce processing of k-means algorithm with FPGA-accelerated computer cluster," in *IEEE Int Conf Application-specific Systems, Architectures and Processors*, June 2014, pp. 9–16.
[45] Y.-k. Choi, J. Cong, Z. Fang, Y. Hao, G. Reinman, and P. Wei, "A quantitative analysis on microarchitectures of modern cpu-fpga platforms," in *Proceedings of the 53rd Annual Design Automation Conference*, ser. DAC '16. New York, NY, USA: ACM, 2016, pp. 109:1–109:6. [Online]. Available: http://doi.acm.org/10.1145/2897937.2897972
[46] J. Stuecheli, B. Blaner, C. Johns, and M. Siegel, "Capi: A coherent accelerator processor interface," *IBM Journal of Research and Development*, vol. 59, no. 1, pp. 7–1, 2015.
[47] T. M. Brewer, "Instruction set innovations for the convey hc-1 computer," *IEEE micro*, vol. 30, no. 2, 2010.
[48] A. V. Hazarika, G. J. S. R. Ram, and E. Jain, "Performance comparision of hadoop and spark engine," in *I-SMAC (IoT in Social, Mobile, Analytics and Cloud)(I-SMAC), 2017 International Conference on*. IEEE, 2017, pp. 671–674.



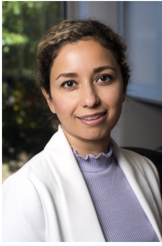
Katayoun Neshatpour is a PhD student at the department of Electrical and Computer Engineering at George Mason University. She is a recipient of the three-year Presidential Fellowship and a 1-year supplemental ECE department scholarship. Advised by Dr. Homayoun and co-advised by Dr. Sasan, her PhD research is on Hardware Acceleration of Big data applications, with a focus on the implementation of several machine learning algorithms in Apache Hadoop and efficient implementation of convolutional neural networks. Katayoun got her Master's degree from Sharif University of Technology, where she worked on the VLSI implementation of a MIMO detector applied to the LTE.

Maria Malik is currently working towards the Ph.D. degree in Electrical and Computer Engineering department, at George Mason University, VA. She has received the M.S. degree in Computer Engineering from the George Washington University, DC and B.E. degree in Computer Engineering from the Center of Advanced Studies in Engineering, Pakistan. Her research interests are in the field of Computer Architecture with the focus of performance characterization and energy optimization of big data applications on the high performance servers and low-power embedded servers, scheduling MapReduce application on microserver, accelerating machine learning kernels, parallel programming languages and parallel computing.

Avesta Sasan received his B.Sc. in Computer Engineering from the University of California Irvine in 2005 with the highest honor (Summa Cum Laude). He then received his M.Sc. and his Ph.D. in Electrical and Computer Engineering from the University of California Irvine in 2006 and 2010 respectively. In 2010, Dr. Sasan joined the Office of CTO in Broadcom Co. working on the physical design and implementation of ARM processors, serving as physical designer, timing signoff specialist, and lead of signal and power integrity signoff in this team. In 2014 Dr. Sasan was recruited by Qualcomm office of VLSI technology. In this role, Dr. Sasan developed different methodology and in-house EDAs for accurate signoff, and analysis of hardened ASIC solutions. Dr. Sasan joined George Mason University in 2016, and he is currently serving as an Associate Professor in the Department of Electrical and Computer Engineering. Dr. Sasan research spans low power design and methodology, hardware security, accelerated computing, approximate computing, near threshold computing, neuromorphic computing, and the Internet of Things (IoT) solutions.

2332-7766 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMSCS.2018.2854787, IEEE Transactions on Multi-Scale Computing Systems



Setareh Rafatirad is an Assistant Professor of the IST department at George Mason University. Prior to joining George Mason, she spent four years as a Research Assistant at UC Irvine. Prior to that, she worked as a software developer on the development of numerous industrial application systems and tools. As a known expert in the field of Data Analytics and Application Design, she has published on a variety of topics related to Big Data, and served on the panel of scientific boards. Setareh received her PhD degree from the Department of Information and Computer Science at the UC Irvine in 2012. She was the recipient of 3-year UC Irvine CS department chair fellowship. She received her MS degree from the Department of Information and Computer Science at the UC Irvine in 2010.

Houman Homayoun is currently working towards the Ph.D. degree in Electrical and Computer Engineering department, at George Mason University, VA. She has received the M.S. degree in Computer Engineering from the George Washington University, DC and B.E. degree in Computer Engineering from the Center of Advanced Studies in Engineering, Pakistan. Her research interests are in the field of Computer Architecture with the focus of performance characterization and energy optimization of big data applications on the high performance servers and low-power embedded servers, scheduling MapReduce application on microserver, accelerating machine learning kernels, parallel programming languages and parallel computing.

2332-7766 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.