J. Parallel Distrib. Comput. 119 (2018) 1–17

Contents lists available at ScienceDirect

J. Parallel Distrib. Comput.

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

Energy-efficient acceleration of MapReduce applications using FPGAs

Katayoun Neshatpour [a,*], Maria Malik [a], Avesta Sasan [a], Setareh Rafatirad [a], Tinoush Mohsenin [b], Hassan Ghasemzadeh [c], Houman Homayoun [a]

[a] Department of Electrical and Computer Engineering, George Mason University, United States
[b] Department of Computer Science and Electrical Engineering, Washington State University, United States
[c] School of Electrical Engineering and Computer Science, Washington State University, United States

HIGHLIGHTS

- MapReduce FPGA acceleration reduces performance and power gap between Xeon and Atom.
- Little core servers are more energy-efficient both before and after acceleration.
- FPGA-accelerated Atom server yields execution times comparable to stand-alone Xeon.
- Cost Analysis suggests replacing big-core with accelerated little core servers.
- FPGA acceleration of MapReduce maintains its benefit when scaled to larger clusters.

ARTICLE INFO

Article history:
Received 9 May 2017
Received in revised form 11 February 2018
Accepted 14 February 2018
Available online 19 March 2018

Keywords:
Machine learning
Hardware+software co-design
Zynq boards
MapReduce
Hadoop
FPGA

ABSTRACT

In this paper, we present a full end-to-end implementation of big data analytics applications in a heterogeneous CPU+FPGA architecture. Selecting the optimal architecture that results in the highest acceleration for big data applications requires an in-depth of each application. Thus, 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 further profile various components of Hadoop MapReduce to identify candidates for hardware acceleration. We accelerate the mapper functions through hardware+software (HW+SW) co-design. Moreover, we study how various parameters at the application (size of input data), system (number of mappers running simultaneously per node and data split size), and architecture (choice of CPU core such as big vs little, e.g., Xeon vs Atom) levels affect the performance and power-efficiency benefits of Hadoop streaming hardware acceleration and the overall performance and energy-efficiency of the system. A promising speedup as well as energy-efficiency gains of up to 8.3× and 15× is achieved, respectively, in an end-to-end Hadoop implementation. Our results show that HW+SW acceleration yields significantly higher speedup on Atom server, reducing the performance gap between little and big cores after the acceleration. On the other hand, HW+SW acceleration reduces the power consumption of Xeon server more significantly, reducing the power gap between little and big cores. Our cost Analysis shows that the FPGA-accelerated Atom server yields execution times that are close to or even lower than stand-alone Xeon server for the studied applications, while reducing the server cost by more than 3×. We confirm the scalability of FPGA acceleration of MapReduce by increasing the data size on 12-node Xeon cluster and show that FPGA acceleration maintains its benefit for larger data sizes on a cluster.

1. Introduction

[1] Emerging big data analytics applications require a significant amount of server computational power. However, these applications share many inherent characteristics that are fundamentally

[1] This work is an extension to a paper we published in IEEE Big Data 2015 [32].

* Corresponding author.

E-mail addresses: kneshatp@gmu.edu (K. Neshatpour), mmalik9@gmu.edu (M. Malik), asasan@gmu.edu (A. Sasan), srafatir@gmu.edu (S. Rafatirad), tinoosh@umbc.edu (T. Mohsenin), hassan@eecs.wsu.edu (H. Ghasemzadeh), hhomayou@gmu.edu (H. Homayoun).

https://doi.org/10.1016/j.jpdc.2018.02.004
0743-7315/

different from traditional desktop, parallel, and scale-out applications [11,30,44]. Big data analytics applications heavily rely on specific deep machine learning and data mining algorithms. They run a complex database software stack with various software components (e.g. Hadoop, Spark, MPI, Hbase, Impala, MySQL, Hive, Shark and MangoDB), which are bounded together with a runtime software system and interact significantly with I/O and OS.

This new set of characteristics 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 as the size of data grows, the semiconductor industry has reached scaling limits and is no longer able to reduce power consumption in new chips. Physical design constraints, such as power and density, have therefore become the dominant limiting factors for scaling out data centers [13,14]. Current server designs, based on commodity homogeneous processors, are not be the most efficient in terms of performance/watt to process big data applications [19].

In other domains, heterogeneous architectures have emerged as a promising solution to enhance energy efficiency by allowing each application to run on a core that matches resource needs more closely than a one size-fits-all processing node [16,40]. This will be just as true for big data applications, if not more so. A heterogeneous chip architecture integrates cores with various micro-architectures (in-order or little, and out-of-order or big) or even instruction set architectures (e.g., Thumb and $\times$86) with on-chip GPU or FPGA accelerators to provide more opportunities for efficient workload mapping so that the application can find a better match among various components to improve power efficiency. In particular, hardware acceleration through specialization, which is enabled by tight integration of CPU core and FPGA logic has received renewed interest in recent years, partially in response to the dark silicon challenge.

As Hadoop MapReduce is a dominant framework in big data analytics, in this paper, we will mainly focus on applications that are developed in this framework. While hardware acceleration has been applied to software implementation of many 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 [17]. Also deploying hardware acceleration methods in a complex environment of big data (such as Hadoop) with various phases of execution such as compression, decompression, sorting, mapping, reduction and shuffling is becoming an even more challenging problem.

Most recent work on hardware acceleration have focused on the implementation of an entire particular machine learning application, or offloading an entire phase of MapReduce to the FPGA hardware [17,25,36,7]. While these approaches provide performance benefit, their implementations require excessive hardware resources and extensive design effort. As an alternative, hardware+software (HW+SW) co-design of an algorithm trades some speedup at a benefit of less hardware and design automation using high level synthesis (HLS) tools.

To understand the potential performance gain of using HW+SW co-design to accelerate analytics applications in MapReduce environment, in this paper we present a full end-to-end implementation of big data analytics applications in a heterogeneous CPU+FPGA architecture, taking into account various communication and computation overhead in the system including the hardware communication overhead such as communication with FPGA and communication among various nodes of a cluster, as well as software computation overhead such as various phases of MapReduce. In [32,33], various applications were studied in an FPGA-accelerated framework. In this paper, we carry out extensive analysis on big and little cores, and how they impact the role of FPGA in the acceleration. Specifically, we compare the results for execution time, power consumption and energy-delay-product (EDP) of both big and little core architecture to show how the FPGA acceleration targets the speed in Little core architecture and the power in Big core architectures. Moreover, we carry out simulations on a 12-node cluster on various data sizes to observe the scalability of FPGA-accelerated architectures.

We assume the applications are fully known, therefore we can find the best possible application-to-core+FPGA match. For mapping of big data analytics applications to FPGA, we are performing the following tasks in this paper:

- Mapping hot regions of various data mining and machine learning algorithms to the FPGA.
- Communication cost analysis of moving hotspot functions to the FPGA.
- Evaluation of HW+SW implementation of the algorithms in an end-to-end MapReduce system in terms of performance and energy efficiency.
- Sensitivity analysis based on the number of mapper slots and input data split size.
- A thorough performance and power comparison between Xeon and Atom server.
- Study the scalability of the FPGA accelerated MapReduce on a 12-node cluster for various input data sizes.

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.
- Performance and power sensitivity to various MapReduce environment system parameters varies significantly before and after acceleration. It also varies across big and little core architectures.
- HW+SW acceleration yields higher speedup on atom server, therefore significantly reducing the performance gap between little and big cores after acceleration.
- HW+SW acceleration yields higher power reduction on Xeon server, therefore significantly reducing the power gap between little and big core architectures after acceleration.
- For some applications (i.e. *K*-means and KNN) the FPGA accelerated Xeon server yields lower power than stand-alone Atom server.
- Atom server is more energy-efficient both before and after acceleration (exhibits lower EDP).
- FPGA acceleration of MapReduce maintains its benefit when scaled to larger clusters.
- Our cost Analysis shows that the FPGA-accelerated Atom server yields execution times that are close to or even lower than stand-alone Xeon server, while reducing both the server cost and the power consumption significantly.

This paper is organized as follows. In Section 2, a background is provided on big data, MapReduce, and Apache Hadoop. In Section 3, the end-to-end system architecture utilized for implementation and analysis purposes is introduced. In Section 4, the acceleration of studied benchmarks is described. Section 5 introduces the analytical assumptions utilized for evaluation of potential speedup and energy-efficiency gain. In Section 7, the results of profiling and hardware implementation of micro kernels and functions within the studied benchmarks are introduced. Section 6 introduces Zynq as the case study for HW+SW acceleration. Section 10 shows the scalability of the FPGA acceleration on a 12-node cluster. Section 11 discusses the related work and Section 12 sums up the conclusions.

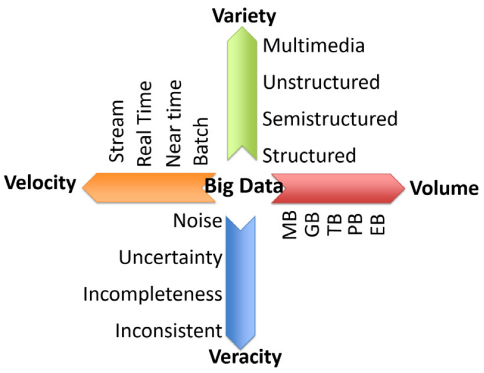


Fig. 1. Illustration of Four Vs of big data.

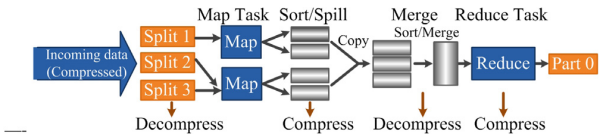


Fig. 2. Hadoop MapReduce: Computational Framework Phases [2].

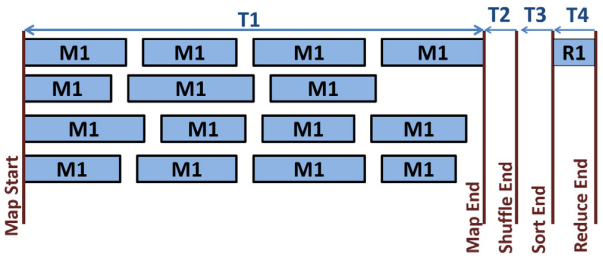


Fig. 3. Timing of various Hadoop phases.

2. Big data and MapReduce framework

The cloud is a new platform that has been used to cost effectively deploy an increasingly wide variety of applications. Vast amount of data is now stored in a few places rather than distributed across a billion isolated computers, therefore it creates opportunity to learn from the aggregated data. The rise of cloud computing and cloud data storage, therefore, has facilitated the emergence of big data applications. Big data applications are characterized by four critical features, referred as the four V, shown in Fig. 1: volume, velocity, variety, and veracity [1]. Big data is inherently large in volume. Velocity refers to how fast the data is generated and to how fast it should be analyzed. In other words, velocity addresses the challenges related to processing data in real-time. Variety refers to the number and diversity of sources of data and databases, such as sensor data, social media, multimedia, text, and much more. Veracity refers to the level of trust, consistency, and completeness in data.

MapReduce is the programming model developed by Google to handle large-scale data analysis. MapReduce consists of map and reduce functions. 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). Hadoop runs the job by breaking it into tasks, i.e., map and reduce tasks. The input data is divided into fixed-size pieces called input splits. Each map task processes a logical split of this data that resides on the Hadoop distributed file system (HDFS). Small input splits yield better load balancing among mappers and reducers at the cost of communication overhead. In order to perform data locality optimization, it is best to run the map task on a node where, input data resides in the HDFS. Thus, the optimal split size for data is the size of an HDFS block (Typically 64 MB, 128 MB, etc.) [45].

The Hadoop MapReduce implementation consists of several phases as depicted in Fig. 2. Compression and decompression are optional phases, which can improve the performance of the system especially for large data sizes.

2.1. Timing

In MapReduce a phase cannot complete before it receives all the data from the previous phase. Thus, the job running time depends on the time it takes to complete all the phases, rather than the duration of individual phases comprising it [8].

While the programming model in MapReduce is comprised of two phases, i.e., map and reduce, its execution model has four phases [4]. The first phase is map in which, the map tasks are executed on the entire input data. The map phase is completely parallel. During this phase, the input data is divided into fixed-size pieces called input splits. One map task is created for each input split, on which the user-defined map function is executed and the <key,value> pairs are generated. In the second phase all <key,value> pairs for a particular key are sent to a single reduce task. In [23], this phase is referred to as all-map-to-all-reduce personalized communication. In MapReduce this phase is called shuffle [10]. In the third phase, the <key,value> pairs are sorted based on the key. In this phase, all the values associated with each key are grouped together. The fourth and final phase is the reduce, in which reduce tasks are applied to the <key,value> pairs of the same key and the final output for that key is generated.

Fig. 3 shows the timing diagram of a MapReduce application with 15 map jobs, 4 mapper slots and one reduce job. In [3] a thorough analysis is carried out on the timing of MapReduce phases. Fig. 3 shows only the timings that are relevant to our work and play a part in the timing analysis after HW+SW acceleration.

As shown in Fig. 3, the map phase starts with the start of the first map task and finishes when the last map task completes its execution. Other phases are also shown in Fig. 3. It should be noted that shuffle starts shortly after the first map, and will not complete until all the map tasks are finished. In this example, shuffle finishes shortly after the last map, i.e., a low-volume shuffle. In case of a high-volume shuffle, the shuffle takes longer to finish. The third phase, i.e., sort, finishes after the shuffle. The reduce phase only starts after all the data is sorted. Upon the completion of the reduce phase, 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 high portion of the execution time across studied machine learning kernels. It should be noted that, even if we are able to accelerate the map phase significantly, still the other phases take up a significant part of the execution time. Thus, they limit the extent to which we are able to accelerate a MapReduce job.

3. System architecture

3.1. Single-node

Fig. 4 shows the system architecture of the single-node, multi-core platform studied in this paper. The multi-node architecture will be discussed in the sequel.

A general-purpose CPU comprises of several identical cores, connected to each other through their shared-memory distributed interconnect. However, for hardware acceleration, we assume each core is extended with a small FPGA fabric. We study how adding on-chip FPGAs to each core would enhance the performance of the multi-core architecture that runs Hadoop MapReduce. While in a general purpose multi-core CPU, mapper/reducer slots are mapped to a single node, in the heterogeneous architecture depicted in Fig. 4, 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 that is accounted for in this work. Thus, each mapper and reducer is 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 Intel Xeon E5. These two types of servers represent two schools of thought in server architecture design: using big Xeon server, which is a conventional approach to designing a high-performance server, and Atom, which is a new trajectory in server design that advocates the use of a low-power core to address the dark silicon challenge facing servers [14,29].

Intel Atom C2758 server deploys 8 processor cores, a two-level cache hierarchy (L1 and L2 cache sizes of 24 KB and 1024 KB, respectively), and an operating frequency of 2.4 GHz with Turbo Boost. Intel Xeon has two socket of six aggressive processor cores per node, three levels of cache hierarchy, private L1 and L2 cache and shared L3 cache. L1, L2 and L3 cache sizes are 32 KB, 256 KB and 15 MB, respectively.

Moreover, each FPGA in Fig. 4 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 established through the Advanced eXtensible Interface (AXI)-interconnect.

AXI is an interface standard through which, different components communicate with each other. The AXI link contains an AXI master, which initiates transactions, and the AXI slave, which responds to the transactions initiated by the master. The data that is 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 memory that is shared between the FPGA and the core. The DMA provides high-bandwidth direct memory access between the AXI-stream and the IP interfaces that are implemented on the FPGA.

3.2. Multi-node

Fig. 5 shows the system architecture of the proposed multi-node cluster. The architecture consists of a homogeneous CPU as the NameNode, which is connected to several DataNodes with a heterogeneous architecture. The architecture of each DataNode is similar to the architecture in Fig. 4.

The NameNode runs the JobTracker and is responsible for the job scheduling between all the DataNodes. It is configured to distribute the computation workloads among the DataNodes. Each DataNode has several mapper and reducer slots. The number of mapper and reducer slots on each DataNode is based on its number of cores. The interconnection between the NameNode and DataNodes is established through a multi channel Gigabit switch to allow high data transfer rates.

4. Benchmark acceleration

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 scheduling and mapping to find the best match. Making wrong scheduling decisions leads to suboptimal performance and negatively impacts power and energy consumption [42].

The primary step to deal with these problems is a comprehensive workload analysis and performance monitoring when running an end-to-end or full system benchmark. To this end, we introduce the architecture of an end-to-end MapReduce implementation.

It is worth noting that MapReduce is best suited for classic machine-learning applications including classification and clustering. For instance, Google Maps utilizes MapReduce to find all roads connecting to a specific intersection, or finding the nearest feature to a given address or current location. Thus, we select four widely used machine-learning application for HW+SW co-design namely, *K*-means, KNN, Naive Bayes and SVM. It should be noted that few works have focused on implementation of neural networks in the MapReduce [26,27]. However, since GPUs are ideal for situations involving applying the same vector/matrix operation across a large number of datasets, most recently they are mainly being used for neural networks. Thus, we limit our experiments to classic machine-learning applications. Subsequently, We characterize each application in order to find out which kernels are the best candidates to be offloaded to the hardware for acceleration.

4.1. Profiling

As a first step, a comprehensive workload analysis and performance monitoring is done for four widely used machine learning kernels. We profile each application using the GNU profiler. We execute the profiling for various input data sizes and different parameters. Table 1 shows the profiling results for selected examples for each application. A detailed description of the results for each application comes in the sequel.

4.1.1. K-means

K-means is a partitioning-based clustering application that partitions n observations into k clusters such that each observation is mapped to the cluster with the closest mean based on specific features. In this paper, the *K*-means application from NU-MineBench [31] is studied.

K-means comprises several kernels. Three dominant ones, which account for a considerable portion of the execution time are *kmeans_clustering*, *find_nearest_point* and *euclid_dist_2*. *Kmean_clustering* is the top function, which calls *find_nearest_point*, which in turn calls *euclid_dist_2*. The timing behavior of the application is highly dependent on input parameters including number of points (N), number of clusters (k), and the feature size (f). Based on Table 1, the time spent in all of the functions increases with the number of points and feature size. Note, that since the functions are nested, the percentage numbers are not adding up to 100%.

4.1.2. KNN

KNN is a pattern recognition algorithm, which finds the k nearest neighbors of a vector among N training vectors based on f features. In order to profile the KNN algorithm, a C-based implementation of KNN is profiled. Table 1 shows the profiling results of KNN classifier for two functions that dominate the execution time, i.e *KNN_classify* and *sq_euclid_dist*. The results show that the time per call increases with the feature size.

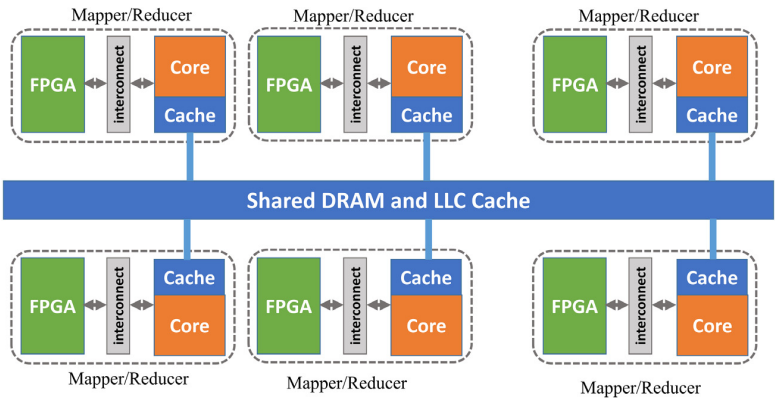


Fig. 4. System architecture of HW accelerator for a single node.

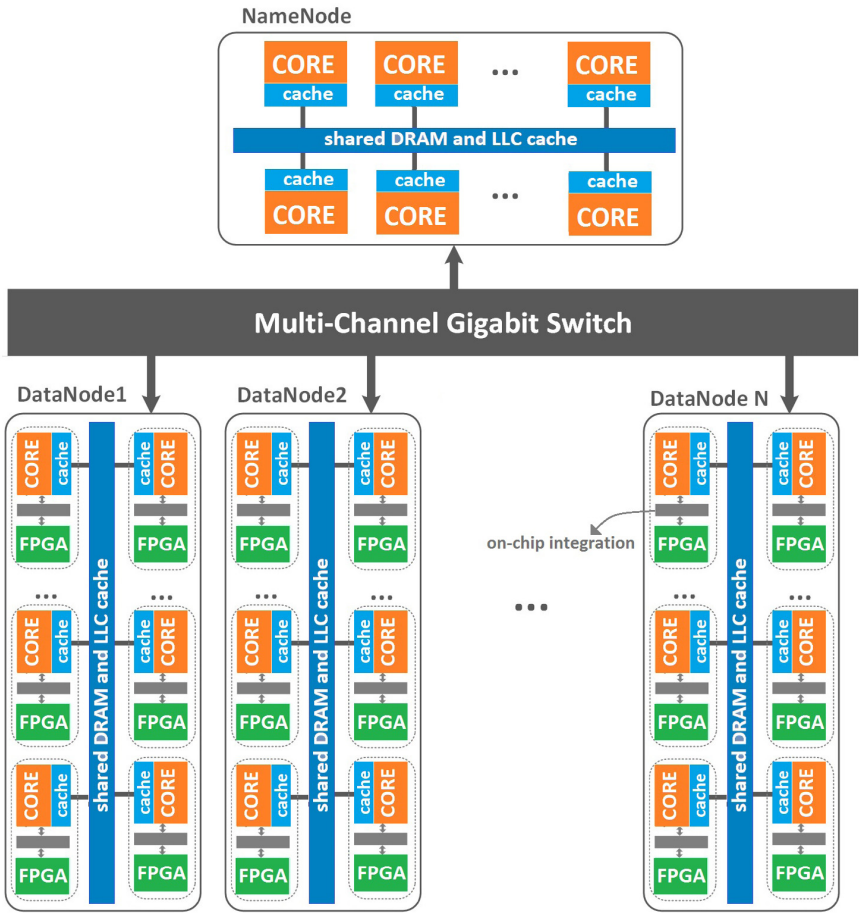


Fig. 5. System architecture for a multi-node cluster.

4.1.3. SVM

SVM is a machine learning algorithm, that is used extensively in data analysis and pattern recognition as non-probabilistic binary linear classifier. We utilize SVM-light C-based code [21] for different datasets. *Sprod_ns*, *sprod_ss* and *solve_dual* are three independent functions, which take up most of the SVM-learn execution time. The first two compute the inner product of sparse vectors, and the latter solves dual problems. Based on the results in Table 1, the number of training documents (N), support vectors (M) and feature size (f) are important factors that influence the execution time.

4.1.4. Naive Bayes

Bayes classifier is another machine learning algorithm, which is used as a probabilistic classifier using strong independent feature model and the Bayesian theorem. A C-based implementation of

Table 1
Profiling results.

Application	Function	Calls	Percentage	Time/call
K-means	Example 1	$N = 100$	$k = 13$	$f = 10$
	K_means_clustering	80	100%	500 μs
	find_nearest_point	10 321	75%	2.91 μs
	euclid_dist_2	89 783	50%	223 ns
	Example 2	$N = 17\,695$	$k = 13$	$f = 18$
	K_means_clustering	80	99.9%	358 ms
	find_nearest_point	17 025 982	84.6%	1.424 μs
	euclid_dist_2	134 110 063	75.7%	162 ns
KNN	Example1		$f = 5$	
	KNN_classify	18 597	97.07%	889.29 μs
	sq_euclid_dist	280 004 358	80.26%	59.39 ns
	Example2		$f = 10$	
	KNN_classify	18 597	99%	1.571 ms
	sq_euclid_dist	280 004 358	88.28%	114.92 ns
SVM	Example 1	$N = 610$	$M = 370$	$f = 9930$
	solve_dual	7136	78.3%	375.56 μs
	sprod_ns	2667 530	8.2% s	105 ns
	sprod_ss	245 091	2.6%	367 ns
	Example 2	$N = 2000$	$M = 879$	$f = 9947$
	solve_dual	7136	91.41%	388.17 μs
	spro_ns	2667 530	8.5% s	97.5 ns
	sprod_ss	245 091	1.3%	163 ns
Naive Bayes	linelength	18 597	13.85%	2.094 μs
	areaunderCurve	18 597	13.48%	1.997 μs
	normDecay	18 597	11.61%	1.687 μs
	abs	9521 459	11.98%	3.3 ns
	nb_classify	18 597	8.99%	1.205 μs

the Naive Bayes was profiled for several examples. *Nb_classify*, *Linelength*, *AreaUnderCurve*, *normDecay* are a number of functions that carry out mathematical operations on the data and *abs* is another function that is called within these functions.

4.2. High-level synthesis of hotspot functions

While a hardware equivalent of a C or C++ function can be realized manually with optimal performance, it can be extremely time consuming for complex functions. In order to speedup this process, high-level synthesis (HLS) is used. HLS is the automated process of transforming a C or C++ code into a register transfer level (RTL) description. The C/C++ language provides constructs to directly access memory. However the synthesis of such structures is a big challenge for HLS tools.

Recent works have addressed the problem of the implementation of dynamic, pointer-based structures [39,46]. Xilinx Vivado HLS tool supports pointer arrays, provided each pointer points to a scalar or an array of scalars. It also supports pointer casting between native C type. However, it does not support array of pointers pointing to additional pointers or general pointer castings [43]. Recent works have addressed the problem of the implementation of dynamic, pointer-based structures [39,46]. Thus, a number of changes were made to the functions that were selected for hardware implementations to synthesize them, while maintaining their functionality.

It should be noted that the hardware implementations of some of the selected functions are dependent on input parameters, which is specific to each example. For the *K*-means applications for instance, the RTL equivalent of the *find_nearest_point* function is highly dependent on the number of points.

Exploration of the dependencies in the code are of high importance for optimization purposes. The dependencies could be at fine granularity, i.e., the dependencies between each iteration of the loops within each function, which can limit the performance benefit gained through pipelining and loop unrolling, given availability of unlimited hardware resources. At a coarse granularity, multiple execution of the same function may be explored for parallelism. If the result of each call of the hotspot function is independent from the result of the previous call, multiple instances of the same function may be instantiated to speed up the overall execution time; however, FPGA transfer bandwidth capacity and hardware resources may become a bottleneck.

Exploitation of parallelism at these levels requires an in-depth analysis of the code, function calls and timing diagram of each application for various inputs. In this paper, we aim to optimize each function exploiting various design techniques such as pipelining and loop unrolling. Further incorporation of inter-function parallelization will be the focus of future work.

Table 2 shows the HLS implementation results of the studied functions on the ZedBoard. All designs are pipelined in order to get the best results. The latency values show the number of clock cycles it takes to produce an output value, which shows the delay of the circuit. The interval values show the number of clock cycles between when the task starts to accept new input data, which is an indicator of the speedup the function can achieve. It should be noted that loop un-rolling and pipelining significantly reduces the interval and thus increases the speed of the accelerated functions. Without them, the throughput of the accelerated functions would have been comparable to the software version.

5. Hardware+software speedup calculation

In case of acceleration of applications in the 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 [6,9,47], however this is not the case for MapReduce. In the map phase 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.

Table 2
HLS implementation results.

Application	Function	Clock [ns]	Latency [cycles]	Interval [cycles]
K-means	K_means_clustering	4.01	686	687
$N = 10, k = 13, f = 10$	find_nearest_point	3.36	2	3
	euclid_dis_2	3.36	1	2
K-means	K_means_clustering	4.01	18 697	18 698
$N = 17\,695, k = 13, f = 18$	find_nearest_point	3.38	2	3
	euclid_dis_2	3.38	1	2
KNN classify	sq_euclid_dist	5.2	154	3
$f = 5$				
KNN classify	sq_euclid_dist	5.2	224	5
$f = 10$				
SVM learn	solve_dual	4.55	105	106
$N = 610, M = 370,$	sprod_ns	3.65	3210	100
$f = 9947$	sprod_ss	3.65	5113	150
SVM learn	solve_dual	4.55	3327	3328
$N = 2000, M = 879$	sprod_ns	3.65	3210	100
$f = 9930$	sprod_ss	3.64	5113	150
	lineLength	3.65	4116	128
	areaUnderCurve	3.65	4134	128
Naive Bayes	normDecay	4.13	4146	128
	abs	3.13	4	1
	nb_classify	4.55	957	25

This significantly increases the benefit of hardware acceleration for MapReduce.

While a thorough investigation of the codes and timing diagrams will help resolve the dependencies and add more parallelism to the implementation, in this paper for the purpose of a general model applicable to all applications, a worse case scenario is considered in which, it is assumed that various functions implemented in the hardware may incur dependencies, thus they are not called simultaneously.

Firstly, we drive the speedup ignoring the overhead. While this is not a realistic assumption given the communication latency in ZedBoard, it provides us with an estimate of the upper-bound speedup that can be achieved. Subsequently, a basic estimation for the transfer time is also incorporated into the calculations for deriving the overall execution time.

5.1. Zero-overhead communication

In Section 4 we derived the amount of time spent in each function in the software implementations. Here, for the purpose of a general model applicable to all applications, it is assumed that various functions implemented in the hardware may not be called simultaneously. The amount of time spent in each function is therefore deducted from the overall execution time to find the time spent in the software. Then, for each function implemented in the hardware, the hardware time per call for each function is calculated as the interval cycles multiplied by the clock period. For each function, the number of times that function is called is multiplied by the hardware time per call to find the hardware time for that function. If more than one function is implemented in the hardware, the total hardware times for all the functions will be added together to get the total hardware times. The final accelerated execution time will be the total software time plus the total hardware time.

Eq. (1) shows the equation used to derive the accelerated time.

$$T_{acc} = T_{orig} - \sum_{i=1}^{n} SW_{i,PC} \times C_i + \sum_{i=1}^{n} HW_{i,PC} \times C_i, \tag{1}$$

where T_{acc} and T_{orig} show the total execution time in the accelerated design and the purely software implementations, respectively. n is the number of accelerated functions, $SW_{i,PC}$ and $HW_{i,PC}$ are the software and hardware time per call for function i, and C_i is the number of calls to function i.

5.2. Modeling the overhead

The assumptions for the calculation of the overhead due to PS–PL communication are described in the sequel; however, these assumptions are used for a first order assessment. Network queueing depth, latency, contention in larger networks and the switching network will have to be addressed for a more detailed assessment.

5.2.1. The PL–PS data transfer overhead

In order to calculate the transfer time, we add the time for communication of the core with the FPGA. Note that the transfer time is device-dependent. Moreover, the size of data that is communicated between the core and the hardware should be compared to the bus bandwidth of the board that is being used. Thus, if the size of transfer data is large, the communication bandwidth between the software and hardware may be potential bottlenecks.

The accelerated time calculated in (1) is modified in (2) to estimate the new accelerated times with the communication overhead.

$$T'_{acc} = T_{acc} + \sum_{i=1}^{n} T_{i,tr} \times C_i, \tag{2}$$

where, T'_{acc} is the total accelerated time and $T_{i,tr}$ is the transmission time for each call of function i. $T_{i,tr}$ is calculated as:

$$T_{i,tr} = \frac{D_i}{BW_{PL,PS}}, \tag{3}$$

where D_i is the size of data being transferred between the PL and PS through each call of the accelerated function, and $BW_{PL,PS}$ is the bandwidth of the data transfer between the PL and PS.

Considering the Zynq architecture, data transfer is done using AXI interconnect. Direct memory access (DMA) is mostly used for larger amount of data to improve the efficiency and reduce CPU intervention. Thus, the contention and the cache misses eventually reduces the effective communication bandwidth.

It should be noted that (2) considers a worse case scenario; however, we are able to reduce the accelerated time by overlapping communication and computation on accelerators.

5.2.2. Communication overhead in Hadoop environment

PCI-Express is used for the communication between the nodes in the system. Based on the number of nodes, the data is transferred

among the various nodes in the system. In this paper, we assume that the entire input data is passed from the master to the slave nodes. Thus, the communication overhead is calculated as follows.

$$T_{ntw} = \frac{\sum_{i=1}^{N} D_{i,ntw}}{BW_{PCI}}, \quad (4)$$

where $D_{i,ntw}$ is the size of data transferred from the master node to node i, BW_{PCI} is bandwidth of the PCI-Express bus and N is the number of nodes.

6. Case study for ZedBoard

AXI is an interface standard through which, different components communicate with each other. The AXI link contains an AXI master, which initiates transactions, and the AXI slave, which responds to the transactions initiated by the master. The are two types of AXI interfaces, AXI memory mapped interface and AXI stream interface.

AXI4 is for memory mapped interfaces and allows burst of up to 256 data transfer cycles with just a single address phase. AXI4-Stream removes the requirement for an address phase altogether and allows unlimited data burst size. AXI4-Stream interfaces and transfers do not have address phases and are therefore not considered to be memory-mapped. Another approach is to build systems that combine AXI4-Stream and AXI memory mapped IP together. Often a DMA engine can be used to move streams in and out of the shared memory. To this end AXI Direct Memory Access (DMA) IPs are utilized in this paper, which provide high-bandwidth direct memory access between the AXI4 memory mapped and AXI4-Stream IP interfaces.

At a 100 MHz clock frequency, data transitions may be realized from AXI4 master to AXI stream slave and AXI stream slave to AXI4 master at data-rates of 400 MBps and 300 MBps, respectively, which are 99.76% and 74.64% of the theoretical bandwidths [28]. These numbers are utilized to find the data transfer overhead between the PL and PS in the Zynq devices.

7. Implementation results

7.1. Acceleration results on the zedboard

Table 3 shows the results of the overall speedup. For each application, various functions were selected for acceleration. The first set of reported speedups is derived based on (1), with zero-overhead. The second set of results includes the overhead. The overhead-included speedup is considerably lower than the zero-overhead speedup, only if the size of data being transferred is large, or the accelerated function is called many times.

For each application, various functions were selected for acceleration. For the K-means application, implementing the top module yields a noticeably high speedup. Table 3 shows that when we select lower-level functions such as *find_nearest_point* and *euclid_dist_2*, the resulting speedup is becoming significantly lower. That is to be expected, since utilization of a dedicated hardware optimized for a specific function will result in a faster design.

Implementation of the *sprod_ns* and *sprod_ss* function for SVM showed that the hardware time for these functions is higher than the software time, which precludes them for being accelerated through hardware. The *solve_dual* function however resulted in lower hardware time, which is thus the only results reported for SVM in Table 3.

For the Naive Bayes algorithm, various functions were selected for hardware implementation. The first results are derived for when only one of the functions were selected and the last result is derived for the case where several functions were selected. The results show that by increasing the number of functions that are moved to the hardware, the speedup increases; however the amount of available hardware should also be considered.

In Table 3, the size of data being transferred during each call of the *k_means_clustering* is considerably high, thus the overhead of the data transmission results in a significant drop in the speedup (64% and 54% drop for feature sizes of 10 and 18, respectively). When the *find_nearest_point function* is accelerated, the size of transferred data during each call is much lower; however, since the number of function calls is higher, we still observe some drop in the speedup (3.2% and 3.6% drop for feature sizes of 10 and 18, respectively).

Table 3 shows that selection of the functions for acceleration is not only concluded based on the implementation on the hardware, but also on the data transfer overhead.

7.2. Acceleration results in Hadoop environment

In this section, we present the speedup results in an end-to-end Hadoop system for offloading time-intensive functions of the studied machine learning kernels to FPGA accelerator. We use Intel Vtune for hotspot analysis of Hadoop MapReduce. Intel VTune is a performance-profiling tool that provides an interface to the processor performance counters [18]. Using Vtune, we analyze the contribution of kernel execution time over the total execution time when running Hadoop.

Fig. 6 shows the common hotspot modules of big data applications on both Intel Atom C2758 and Xeon E5, for different number of mappers. Application kernel represents the computation part to perform the task such as K-means, KNN, etc. Libz performs the data compression and decompression tasks for the Hadoop workload.

Amdahl's law is used to calculate the overall speedup on an end-to-end Hadoop system, based on the speedup results from Table 3 and Hadoop hotspot analysis in Fig. 6.

Fig. 7 shows the achievable acceleration of the Hadoop system for selected examples in Table 3 for an architecture with four mapper slots. Results show that the speedup of each application through HW+SW acceleration is translated into a lower speedup on the end-to-end Hadoop system. For instance, while the acceleration of the K-means yields a speedup of the order of 312× with zero overhead, the speedup drops to 146× with the data transfer overhead, and 2.78× and 2.29× on Hadoop platform with Atom and Xeon, respectively. The final speedup is greatly affected by the fraction of time the kernel execution takes to run in the Hadoop environment. In other words, even if we are able to significantly accelerate the kernel through HW+SW co-design, the overall acceleration on the Hadoop platform is insignificant if most of the time is spent on operations other than the kernel, including data compression and transfers.

7.3. Power and energy-delay product

An important benefit of offloading applications to dedicated hardware is enhancing power efficiency. General-purpose CPUs such as Atom and Xeon are not designed to provide maximum efficiency for every application. Accelerators can help improve the efficiency by not only speeding up execution time, but also executing the task with just enough required hardware resources.

The power values were calculated with the same methodology as the one used to calculate the execution time in Section 5. Power measurement for each application is done using picoScope digital oscilloscope for the ARM and FPGA board (the board idle power was deducted from the power readings value when running application to estimate core power). We measured the power by measuring the current flowing to core and multiplying that by the core voltage. To measure the current, we measured the voltage

Table 3
HLS implementation results.

Application	Accelerated function	Zero-overhead Speedup	Overhead-included Speedup
K-means	$N = 100, k = 13, f = 10$		
	k_means_clustering	181.5	65.08
	find_nearest_point	3.96	3.83
	euclid_disc_2	1.94	1.94
	$N = 17\,695, k = 13, f = 18$		
	k_means_clustering	312.5	146.83
	find_nearest_point	5.89	5.77
	euclid_dist_2	3.64	3.64
KNN	$f = 5$		
	sq_euclid_dist	2.44	1.92
	$f = 10$		
	sq_euclid_dist	3.15	2.37
SVM	$N = 2000, M = 879\, f = 9947$		
	solve_dual	4.03	4.03
	$N = 610, M = 370\, f = 9930$		
	solve_dual	8.23	8.23
Naive Bayes	lineLength	1.126	1.126
	areaUnderCurve	1.138	1.138
	normDecay	1.091	1.091
	nb_classify	1.093	1.093
	mh_abs	1.0071	1.0071
	(lineLength, areaUnderCurve, normDecay, nb_classify)	1.629	1.629

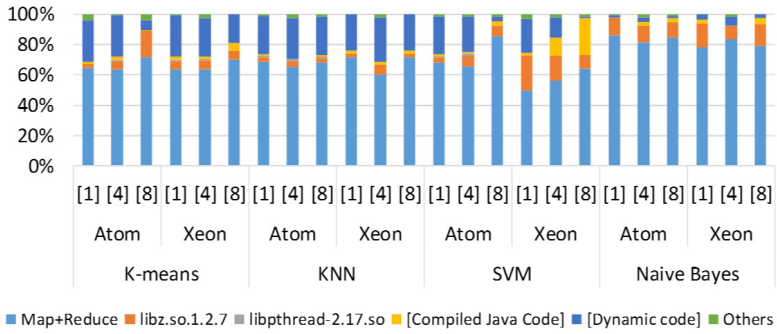


Fig. 6. Hotspot analysis before acceleration.

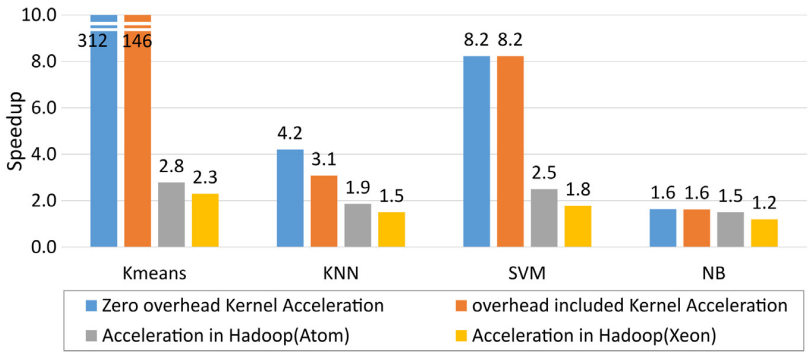


Fig. 7. Acceleration on a full-blown system with four mapper slots.

drop across the test points provided on the board divided by the resistance around those points.

Assuming a uniform distribution of energy over the execution time, the energy dissipation of the accelerated regions was replaced with that of the FPGA to get the energy of the accelerated kernel. Wattsup pro meter was used for power reading on the end-to-end Hadoop system running on Xeon and Atom servers; however, only a fraction of the total energy is due to the kernel, which is substituted with the energy of accelerated kernel. By averaging the resulting Hadoop energy over the new execution time, the power consumption values were calculated.

Table 4 shows the power and energy-delay-product (EDP) results for four mappers on Xeon and Atom. The initial power numbers refer to the power values on Hadoop before acceleration and the accelerated power refer to power number after acceleration. Power ratio shows the ratio of power before the acceleration to the power after the acceleration. EDP shows the energy delay product.

Based on Table 4, power is reduced by up to 3.7× in the studied applications. Moreover, since both the execution time and the power has been reduced through the acceleration, EDP is significantly reduced by up to 15.21×.

8. Sensitivity analysis

8.1. Number of mapper slots

An important factor in determining the performance of an analytics application running in Hadoop MapReduce environment is the number of mapper and reducer slots. The optimal number of mappers and reducers aims to create a balance among the CPU computing power and the amount of data that is transported across nodes. In this section, we evaluate FPGA acceleration results using different number of mappers for running the Hadoop with 1, 4 and 8 mapper slots.

Fig. 6 shows the results of the hotspot analysis for different number of mappers. Fig. 8 shows the FPGA acceleration results using different number of mapper slots.

Results show that for Atom, *K*-means, KNN and SVM yield almost the same execution time for different number of mapper slots and for the Naive Bayes, four mapper slots yield the best results. When Xeon server runs the master node, for KNN and SVM, four mapper slots yield the best results.

While increasing the number of mapper slots in the architecture allows the exploitation of more levels of parallelism, the communication overhead limits the range of achievable speedup. For instance, Fig. 8 shows that increasing the number of mapper slots to four enhances the performance; however, further increase has no or negative effect on the execution time. However, the optimal configuration is highly dependent on the application type, the architecture of the master node, the hotspot characteristics of the application on the Hadoop framework, size of input data splits and the implementation.

8.2. Size of data

In the MapReduce platform, the input data is split into a number of input splits. Each map task processes a logical split of data. The splits of data go to the kernel. Thus, the size of the data that goes to the accelerated kernels is the size of the input splits (not to be confused with the size of input data).

It should be noted that the number of times each function is called within each run of the algorithm, and the fraction of the execution time devoted to that function is dependent on the size of input data. Thus, we conducted the data size sensitivity analysis of studied machine learning applications on the Zynq platform to find out the trend of speedup changes with respect to the input data size. For each application, the profiler calculated the fraction of time spent for each accelerated function and the corresponding speedup was calculated.

Table 5 shows the execution time after acceleration and the speedup for different data sizes for all applications. Table 5 shows that as expected, the execution time increases with the increase in data size. Moreover, the data size has little impact on the achievable speedup, which is due to the overhead for the transfer of data through the switching network and PCI-express.

Fig. 9 shows the speedup results for a studied range of data sizes. Since the execution of some of the applications including the Naive Bayes on small data sizes resulted in very low execution times, (less than 0.1 s), the profiling results were not reliable and thus not reported.

As the results show, the input data size have a significant effect on the speedup in some applications. In case of the *K*-means algorithm, the speedup increases significantly with the size of input data. The size of data does not have much effect on the speedup of Naive Bayes and KNN. Finally, the speedup of the SVM algorithm decreases as the data size increases in the SVM algorithm. The different trend observed in these applications for small sizes is due to both the PS–PL overhead and change in the fraction of execution time devoted to different accelerated functions. However, as input data size increases (beyond 10 MB for these applications), the speedup values start to converge.

9. Performance and cost analysis

9.1. Power and performance comparison of big and little core architecture

We studied two very distinct server microarchitectures; a high performance big Xeon core and a low power embedded-like little Atom core. These two types of servers represent two schools of thought on server architecture design: using big core like Xeon, which is a conventional approach to designing a high-performance server, and the Atom, which is a new trajectory in server design that advocates the use of a low-power core to address the dark silicon challenge facing servers.

Based on the results from Fig. 7, the range of speedup achieved through HW+SW co-design is the higher for Atom considering all the applications. This is due to the fact that for the Atom server, a larger part the application is being accelerated at a higher rate. To compare the overall execution time, Fig. 10 shows the execution time before and after the acceleration for Atom and Xeon. The figure shows that the overall execution time is lower on the architecture in which, HDFS runs on the high-end cores (Xeon). However the acceleration reduces the execution time of Atom at a higher rate. Thus FPGA acceleration, reduces the performance gap between Atom and Xeon.

The results show that the FPGA-accelerated Atom server outperform the non-accelerated Xeon server for most of the applications. This is due to the fact that on the one hand, for these applications the map phase accounts for a higher portion of the execution time on Atom, and on the other hand, the map functions on slower Atom server are accelerated at a higher rate.

Fig. 11 shows the power consumption of Atom and Xeon server before and after the acceleration. The results show that while the power consumption is higher on the Xeon server, the rate of power reduction due to FPGA acceleration is higher on the Xeon server too, thus reducing the power gap between Atom and Xeon. For some applications (i.e., *K*-means and KNN) the power consumption of FPGA-accelerated Xeon server is lower that the non-accelerates Atom server. This is to be expected, as Atom uses small cores, designed to consume low power. However, the power reduction through hardware acceleration is lower for the low-end server (Atom), since it already consumes lower power.

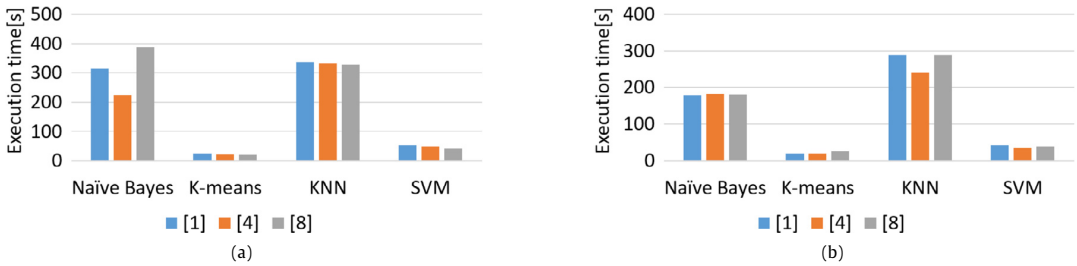


Fig. 8. Execution time after acceleration for various number of mapper slots on Atom (a) and Xeon (b).

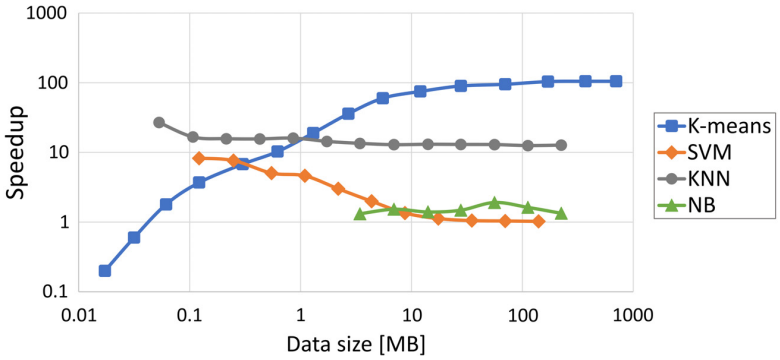


Fig. 9. Kernel acceleration for different input data sizes.

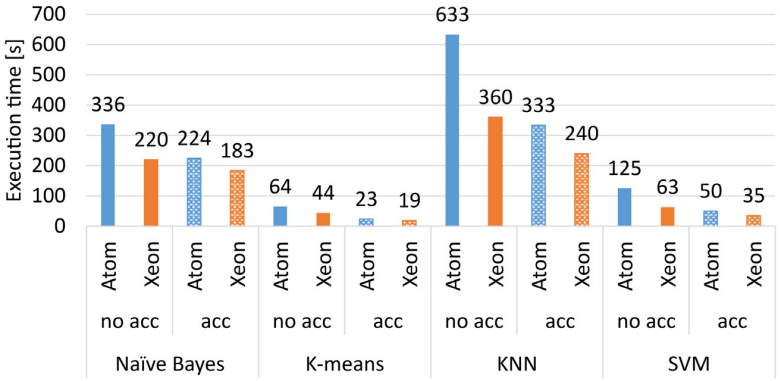


Fig. 10. Execution time comparison of Atom and Xeon.

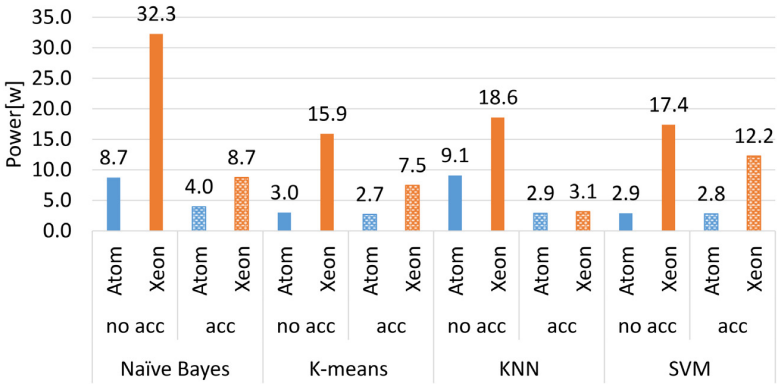


Fig. 11. Power comparison of Atom and Xeon.

Table 4
Power and energy analysis for four mapper slots.

Application	ini power [W]	acc power [W]	ini EDP [W s^2]	acc EDP [W s^2]	power ratio	EDP ratio
			Xeon			
K-means	15.88	7.45	42939.52	2823.1	2.13	15.21
KNN	18.59	9.07	22330.06	3853.57	2.05	5.79
SVM	17.37	12.24	13615.73	4601.9	1.42	2.96
NB	32.26	8.72	18580.43	2316.3	3.7	8.02
			Atom			
K-means	2.99	2.68	5412.17	680.84	1.11	7.95
KNN	3.14	2.87	14912.68	4283.73	1.09	3.48
SVM	2.87	2.77	5062.68	862.09	1.03	5.87
NB	4.26	3.96	5103.49	2242.03	1.07	2.28

Table 5
Execution time for different data sizes.

	Execution time [s]			Speedup		
Data input	284 KB	128 MB	2 GB	284 KB	128 MB	2 GB
K-means	19.20	181.60	303.26	2.708	2.708	2.707
Data input	172 K	4 MB	16 MB	172 K	4 MB	16 MB
KNN	19.91	132.90	504.62	1.740	1.740	1.740
Data input	100 MB	2 GB	10 GB	100 MB	2 GB	10 GB
SVM	18.93	85.71	337.31	1.479	1.479	1.478
Data input	100 MB	2 GB	7 GB	100 MB	2 GB	7 GB
Naive Bayes	16.62	48.48	104.57	1.444	1.444	1.443

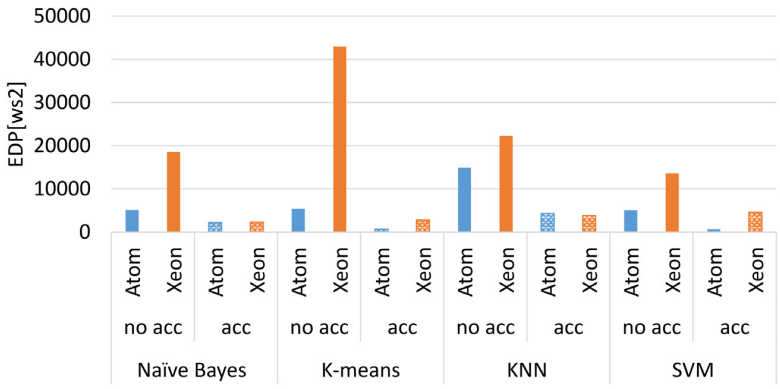


Fig. 12. EDP time comparison of Atom and Xeon.

Fig. 11 shows the EDP of Atom and Xeon server before and after the acceleration. The results show that while the energy efficiency is enhanced more significantly for the high-end server (Xeon), both stand-alone Atom server and FPGA-accelerated Atom server still yield lower EDP and exhibit enhanced energy-efficiency (see Fig. 12).

The comparison of Intel Atom and Xeon shows how the FPGA acceleration targets the speed in Little core architecture and the power in Big core architectures. Moreover, the FPGA-accelerated little core architecture is the most energy-efficient choice for MapReduce acceleration, with its execution time comparable to (and sometimes even lower than) that of Xeon. This finding is significant in making architectural decisions while designing an efficient platform for MapReduce.

9.2. Cost analysis

Fig. 13 shows the capital cost of studied architectures normalized to the cost of Atom server. We utilize conventional costs for various components with Xeon and a low-end FPGA (Artrix-7) having 4.1× and 30% of the cost of an Atom server. The figure shows

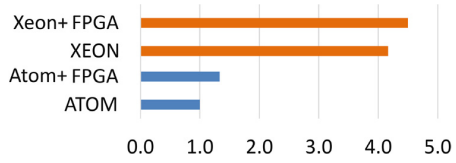


Fig. 13. Cost comparison for various architectures.

that while execution time of applications on the — Xeon+FPGA platform is the lowest, it is not the most cost-efficient solution. On the other hand, adding FPGAs to Atom server is a cost effective solution to get enhanced performance without having to move to expensive servers like Xeon. The FPGA-accelerated Atom server yields execution times that are close to or lower than stand-alone Xeon server for the studied applications, while reducing the server cost by more than 3×. However, if the main optimization goal is the execution time, accelerating Xeon server is the best solution, albeit not cost-efficient.

10. 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 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 (2.60 GHz) 8 core CPUs allowing up to 176 mapper/reducer slots. We execute the applications for various data sizes. The HDFS block size is set to 128 MB. 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 mapper 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, accelerates the MapReduce applications running on a cluster, with a speedup comparable to that on a single-node.

Moreover, as shown in Table 6, the execution times before and after the acceleration change 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, which is the contribution of map time to the total time.

Moreover, it is interesting to observe the speedup for various input data sizes. Fig. 15 shows the speedup for the studied data sizes. The results shows variation of up to 40% in the speedup for the some applications, (i.e., SVM and KNN), and monotonic change in the speedup for others (i.e., Naive Bayes and *K*-means). Regardless, all the applications benefit from the FPGA acceleration for large data sizes.

Fig. 14 shows the execution time both before and after the acceleration for the studied applications, which shows that on the 12-node cluster, the gap between the execution time of the acceleration applications, and the non-accelerated ones does not change significantly over various range of data. By extending the results from one server to a multi-node cluster and experimenting with large data size, we show that for some applications, the FPGA acceleration of MapReduce is more significant, when the size of data increases. This is due the data parallel nature of the MapReduce applications and promise extensive benefits in the world of Big Data, with its ever-increasing demand for processing large data sizes.

11. Related work

The performance and bottlenecks of Hadoop MapReduce have been extensively studied in recent work [17,20,41,49]. To enhance the performance of MapReduce and based on the bottlenecks found for various applications, hardware accelerators are finding their ways in system architectures [15,38].

In MARS [15], a MapReduce framework on an NVIDIA G80 GPU was developed in which the GPU's computational power was harnessed for the MapReduce. In this platform the programming complexity of the GPU was hidden behind the simple and familiar MapReduce interface. The results yield a 16× faster speed than a quad-core CPU-based implementation for six common web applications. In [38], GPU MapReduce (GPMR) is presented as a standalone MapReduce library that modifies the typical MapReduce tasks to fit into GPMR and leverage the power of GPU clusters for large-scale computing. 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 [37]. Alternatively, FPGAs have shown to be more energy efficient, specifically for handheld devices where prolonged battery life is one of the most important design goals [12,37].

Given the industry trend (e.g., Microsoft Catapult) it seems GPU is not the accelerator of choice for cloud scale computing running diverse type of workloads. For domain specific applications such as neural network, however, GPU remains a competitive choice. 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 [24], 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. Microsoft Catapult has built a composable, reconfigurable fabric to accelerate portions of large-scale software services, which consists of 6 × 8 2-D torus of high-end Stratix V FPGAs embedded into a half-rack of 48 machines. One FPGA is placed into each server, accessible through PCIe, and wired directly to other FPGAs with pairs of 10 Gb SAS cables [35]. 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. Another example is Zynq-7000 SoC platform, which integrates ARM cores with Xilinx FPGAs. In [36], a MapReduce framework on FPGA (FPMR) is described in which programming abstraction, hardware architecture and basic building blocks are introduced for RankBoost application along with an on-chip processor scheduler that maximizes the utilization of computation resources and enhances the load balancing. This platform is utilized to compare the performance of RankBoost application with respect to its full software implementation. In [48], 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. The FPGA-based workers are shown to offer ×10 faster task processing. In [5], a MapReduce programming model is evaluated that exploits the computing capacity present in a cluster of nodes equipped with hardware accelerators. They use the cluster of Cell BE processors to accelerate mapper functions for the AES encryption as a data-intensive application and Pi number estimator as a CPU-intensive application. They show that benefits for CPU-intensive applications are remarkable; however, the communication and synchronization overheads can significantly degrade the performance benefits of hardware accelerators in data-intensive applications.

In [17], 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. The Terasort benchmark is utilized to evaluate the proposed architecture. In Zcluster [25], hardware acceleration of FIR is explored through an eight-salve Zynq-based MapReduce architecture. Zcluster is implemented for a standard

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	70.19	1.32
5	40	401	52.62	319.45	1.26
10	71	583	41.17	490.23	1.19
20	69	1256	46.02	1032.60	1.22
100	71	6031	47.55	4517.34	1.16
			K-means		
1	8	20	85.21	3.12	6.42
5	40	49	86.88	7.15	6.85
10	78	62	87.16	8.74	7.1
20	78	87	88.40	10.96	7.93
100	85	383	88.65	48.27	7.93
			KNN		
1	8	245	87.35	57.42	4.27
5	40	577	78.86	178.17	3.24
10	69	872	393.41		2.22
20	80	1297	74.50	450.26	2.88
100	82	4849	78.67	1504.91	3.22
			SVM		
1	8	18	83.33	4.82	3.73
5	40	21	71.43	7.82	2.68
10	71	37	81.08	10.65	3.48
20	76	51	70.59	19.37	2.63
100	77	173	88.44	38.59	4.48

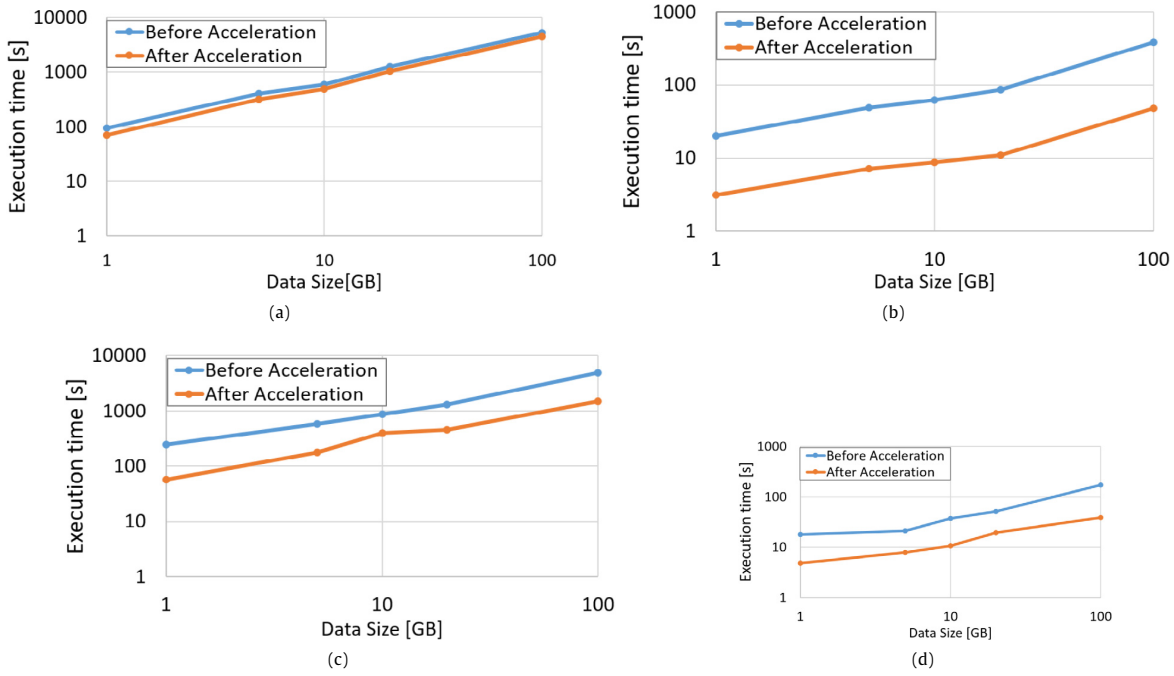


Fig. 14. Execution time before and after the acceleration on a 12-node cluster for (a) Naive Bayes, (b) *K*-means, (c) KNN and (d) SVM.

FIR filter to show the benefits gained through hardware acceleration in the MapReduce framework, where the entire low-pass filter is implemented on the FPGA. In [22], a configurable hardware accelerator is used to speed up the processing of reduce tasks in cloud computing applications on the MapReduce framework. The accelerator is utilized to carry out the reduce tasks. The MapReduce accelerator has been implemented and mapped to a multi-core FPGA with embedded ARM processors. They showed up to 1.8× system speedup of the MapReduce applications. In [7], a detailed MapReduce implementation of the *K*-means application is presented. Which was shown to enhance the speedup of software implementation of *K*-means.

In recent years, several FPGA-based heterogeneous accelerators, have been proposed for big data analytics and cloud computing and in particular for MapReduce programming model. However, these efforts mainly attempt to accelerate a particular MapReduce application and deploy it on a specific architecture that fits well the performance and power requirements of the

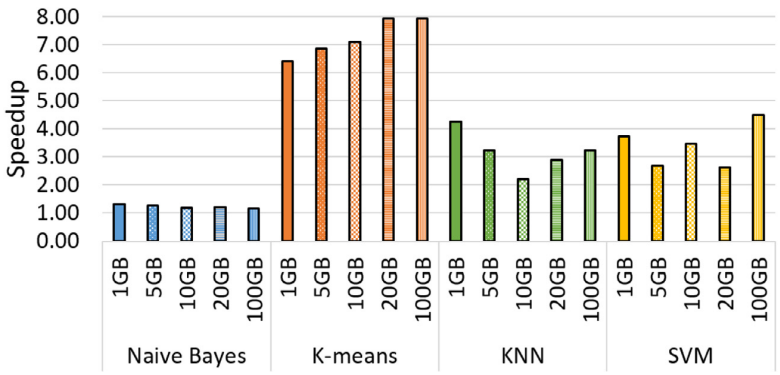


Fig. 15. Speedup of the execution for various data sizes on a 12-node cluster.

application. Given the diversity of architectures presented, the important research question is which architecture is better suited and more cost-efficient to meet the performance, power and energy-efficiency requirements of a wide and diverse range of MapReduce applications. There has been no prior effort on performing design space exploration to find the choice of hardware accelerator architecture that can be deployed in cloud for MapReduce analytics applications. Our experiments answer the questions of what is the role of processor after acceleration in a heterogeneous architecture; whether a high-end server is most suited to run big data applications or a low-end server equipped with FPGA provides sufficient performance with lower power consumption and capital cost?

12. Conclusions

In this paper we presented an end-to-end implementation of big data analytics applications in a heterogeneous CPU+FPGA architecture. We developed the MapReduce parallel implementation of K-means, KNN, SVM and naive Bayes in a Hadoop MapReduce environment and profiled various phases of Hadoop execution on Atom and Xeon architectures to understand the breakdown of execution time on various microarchitectures. We accelerated the mapper functions through HW+SW co-design on the Zynq FPGA platform. The results show promising speedups as well as energy-efficiency gains of up to 8.2× and 15×, respectively for large data sizes. We further studied how application, system, and architecture level parameters such as the choice of CPU (big vs little), the number of mapper slots, and the data split size affect the performance and power-efficiency benefits of Hadoop Streaming hardware acceleration. Our analysis concluded that power and performance sensitivity to various MapReduce environment system parameters vary significantly before and after acceleration. They also vary across big and little core architectures. The results show that HW+SW acceleration yields higher speedup on Atom server, therefore significantly reducing the performance gap between little and big cores after acceleration. The results show that the FPGA-accelerated Atom server outperforms the non-accelerated Xeon server for most of the applications. Moreover the results show that while the power consumption is higher on the Xeon server, the rate of power reduction due to FPGA acceleration is higher on the Xeon server too, thus reducing the power gap between Atom and Xeon. For some applications. (i.e., K-means and KNN) the power consumption of FPGA-accelerated Xeon server is lower that the non-accelerates Atom server. Based on the results and capital cost analysis, when the main optimization goal is the execution time, accelerating Xeon server is the best solution, albeit not cost-efficient, since FPGA-accelerated Atom server yields execution times that are close to or lower than stand-alone Xeon server for the studied applications, while reducing the server cost by more than 3×. To confirm the scalability of FPGA acceleration for MapReduce, we study a 12-node Xeon cluster and increase the size of data to observer the range of speedup. While a small variation is observed in the speedup of some applications, all the applications benefit from FPGA acceleration at large data sizes. Moreover, for some applications, the speedup is higher for larger data sizes.

References

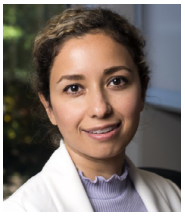
[1] Accelerating hadoop applications using intel quichassist technology, http://www.intel.com/content/dam/www/public/us/en/documents/solution-briefs/accelerating-hadoop-applications-brief.pdf.
[2] Accelerating hadoop applications using intel quickassist technology http://www.intel.com/content/dam/www/public/us/en/documents/solution-briefs/accelerating-hadoop-applications-brief.pdf Accessed: 2014-11-30.
[3] F. Ahmad, S.T. Chakradhar, A. Raghunathan, T.N. Vijaykumar, optimizing MapReduce on heterogeneous clusters, in: Proceedings of the Seventeenth International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS, pp. 61–74.
[4] F. Ahmad, S. Lee, M. Thottethodi, T.N. Vijaykumar, MapReduce with communication overlap (MaRCO), J. Parallel Distrib. Comput. (2013) 608–620.
[5] Y. Becerra, V. Beltran, D. Carrera, M. Gonzalez, J. Torres, E. Ayguade, Speeding up distributed mapreduce applications using hardware accelerators, in: Parallel Processing, 2009. ICPP '09. International Conference on, 2009, pp. 42–49.
[6] P. Eles, 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, DATE '98, IEEE Computer Society, Washington, DC, USA, 1998, pp. 132–139 http://dl.acm.org/citation.cfm?id=368058.368119.
[7] Y.-M. Choi, H.-H. So, Map-reduce processing of k-means algorithm with FPGA-accelerated computer cluster, IEEE ASAP (2014) 9–16.
[8] M. Chowdhury, M. Zaharia, J. Ma, M.I. Jordan, I. Stoica, Managing data transfers in computer clusters with orchestra, in: Proceedings of the ACM SIGCOMM 2011 Conference, 2011, pp. 98–109.
[9] L.A. Cortes, L. Alej, R. Corts, P. Eles, Z. Peng, A survey on hardware/software codesign representation models, 1999.
[10] J. Dean, S. Ghemawat, MapReduce: simplified data processing on large clusters, in: Proc. Conf Symp Operation Systems Design and Implementation, 2004.
[11] M. Ferdman, A. Adileh, O. Kocberber, S. Volos, M. Alisafaee, D. Jevdjic, C. Kaynak, A.D. Popescu, A. Ailamaki, B. Falsafi, Clearing the clouds: A study of emerging scale-out workloads on modern hardware, SIGPLAN Not. 47 (4) (2012) 37–48.
[12] J. Fowers, G. Brown, P. Cooke, G. Stitt, A performance and energy comparison of FPGAs, GPUs, and multicores for sliding-window applications, in: Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays, FPGA '12, 2012, New York, NY, USA pp. 47–56.
[13] A. Gutierrez, M. Cieslak, B. Giridhar, R.G. Dreslinski, L. Ceze, T. Mudge, Integrated 3d-stacked server designs for increasing physical density of key-value stores, in: Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS '14, ACM, 2014, pp. 485–498.
[14] N. e. Hardavellas, Toward dark silicon in servers, IEEE Micro 31 (2011) 6–15.
[15] B. He, W. Fang, Q. Luo, N.K. Govindaraju, T. Wang, Mars: A MapReduce framework on graphics processors, in: Proc Int Conf Parallel Architectures and Compilation Techniques, 2008, pp. 260–269.

[16] H. Homayoun, V. Kontorinis, A. Shayan, T.W. Lin, D.M. Tullsen, Dynamically heterogeneous cores through 3D resource pooling, in: IEEE International Symposium on High-Performance Comp Architecture, 2012, pp. 1–12.
[17] T. Honjo, K. Oikawa, Hardware acceleration of hadoop mapreduce, in: IEEE Int. Conf. Big Data, 2013, pp. 118–124.
[18] Intel vtune amplifier xe performance profiler http://software.intel.com/en-us/articles/intel-vtune-amplifier-xe/, Accessed: 2014-11-30.
[19] V. Janapa Reddi, B.C. Lee, T. Chilimbi, K. Vaid, Web search using mobile cores: Quantifying and mitigating the price of efficiency, ACM SIGARCH Comput. Archit. News 38 (3) (2010) 314–325.
[20] D. Jiang, B.C. Ooi, L. Shi, S. Wu, The performance of MapReduce: An in-depth study, Proc. VLDB Endow. 3 (1–2) (2010) 472–483.
[21] T. Joachims, Making large-Scale SVM learning practical, in: B. Schölkopf, C. Burges, A. Smola (Eds.), Advances in Kernel Methods - Support Vector Learning, MIT Press, Cambridge, MA, 1999, pp. 169–184.
[22] C. Kachris, G.C. Sirakoulis, D. Soudris, A configurable mapreduce accelerator for multi-core FPGAs (abstract Only), in: Proc ACM/SIGDA Intl Symp FPGAs, 2014 241–241.
[23] V. Kumar, A. Grama, A. Gupta, G. Karpis, Introduction to parallel computing: Design and analysis of parallel algorithms, 1994.
[24] A. Kundu, K. Banerjee, N. Mellempudi, D. Mudigere, D. Das, B. Kaul, P. Dubey, Ternary residual networks, 2017, arXiv preprint arXiv:1707.04679.
[25] Z. Lin, P. Chow, ZCluster: A Zynq-based Hadoop cluster, 2013 FTP, 2013, pp. 450–453.
[26] Y. Liu, J. Yang, Y. Huang, L. Xu, S. Li, M. Qi, Mapreduce based parallel neural networks in enabling large scale machine learning, Comput. Intell. Neurosci. 2015 (2015) 1.
[27] Z. Liu, H. Li, G. Miao, Mapreduce-based backpropagation neural network over large scale mobile data, in: Natural Computation, ICNC, 2010 Sixth International Conference on, Vol. 4, IEEE, 2010, pp. 1726–1730.
[28] LogiCore AXI DMA v7.1, Product Guide for Vivado Design Suite, 2013.
[29] M. Malik, K. Neshatpour, T. Mohsenin, A. Sasan, 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] M. Malik, K. Neshatpour, S. Rafatirad, H. Homayoun, Hadoop workloads characterization for performance and energy efficiency optimizations on microservers, IEEE Trans. Multi-Scale Comput. Syst. (2017).
[31] e.a. Narayanan, MineBench: A benchmark suite for data mining workloads, in: IEEE Int Symp Workload Characterization, 2006, pp. 182–188.
[32] K. Neshatpour, M. Malik, M.A. Ghodrat, A. Sasan, H. Homayoun, Energy-efficient acceleration of big data analytics applications using fpgas, in: Big Data (Big Data), 2015 IEEE International Conference on, IEEE, 2015, pp. 115–123.
[33] K. Neshatpour, A. Sasan, H. Homayoun, Big data analytics on heterogeneous accelerator architectures, in: 2016 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS), 2016, pp. 1–3.
[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. Putnam, A. Caulfield, E. Chung, D. Chiou, K. Constantinides, J. Demme, H. Esmaeilzadeh, J. Fowers, G. Gopal, J. Gray, M. Haselman, S. Hauck, S. Heil, A. Hormati, J.-Y. Kim, S. Lanka, J. Larus, E. Peterson, S. Pope, A. Smith, J. Thong, P. Xiao, D. Burger, A reconfigurable fabric for accelerating large-scale datacenter services, in: Computer Architecture (ISCA), 2014 ACM/IEEE 41st International Symposium on, 2014, pp. 13–24.
[36] Y.e. a. Shan, FPMR: mapreduce framework on fpga, in: Proc. ACM/SIGDA Int Symp FPGA, pp. 93–102.
[37] L. Stolz, H. Endt, M. Vaaraniemi, D. Zehe, W. Stechele, Energy consumption of graphic processing units with respect to automotive use-cases, in: Energy Aware Computing, ICEAC, 2010 International Conference on, 2010, pp. 1–4.
[38] J.A. Stuart, J.D. Owens, Multi-GPU MapReduce on GPU clusters, in: IPDPS 2011, 2011, pp. 1068–1079.
[39] L. Smria, K. Sato, G.D. Micheli, Synthesis of hardware models in C with pointers and complex data structures, IEEE TVLSI 9 (6) (2001) 743–756.
[40] M.K. Tavana, A. Kulkarni, A. Rahimi, T. Mohsenin, H. Homayoun, Energy-efficient mapping of biomedical applications on domain-specific accelerator under process variation, in: 2014 IEEE/ACM International Symposium on Low Power Electronics and Design, ISLPED, pp. 275–278, 2014.
[41] C. Tian, H. Zhou, Y. He, L. Zha, A dynamic mapreduce scheduler for heterogeneous workloads, 2009, GCC 2009, pp. 218–224.
[42] e. a. Van Craeynest, Scheduling heterogeneous multi-cores through performance impact estimation, PIE in: ISCA 2012, 2012, pp. 213–224.
[43] Vivado Design Suite User Guide: High-Level Synthesis, http://www.xilinx.com/support/documentation/sw_manuals/xilinx2013_2/ug902-vivado-high-level-synthesis.pdf..
[44] L. Wang, J. Zhan, C. Luo, Y. Zhu, Q. Yang, Y. He, W. Gao, Z. Jia, Y. Shi, S. Zhang, C. Zheng, G. Lu, K. Zhan, X. Li, B. Qiu, BigDataBench: A big data benchmark suite from internet services, in: High Performance Computer Architecture, HPCA, 2014 IEEE 20th International Symposium on, 2014, pp. 488–499.
[45] T. White, Hadoop: The Definitive Guide, first ed., O'Reilly Media, Inc., 2009.
[46] F. Winterstein, S. Bayliss, G. Constantinides, High-level synthesis of dynamic data structures: A case study using Vivado HLS, in: 2013 FTP, 2013, pp. 362–365.
[47] W.H. Wolf, Hardware-software co-design of embedded systems, Proc. IEEE (1994) 967–989.
[48] D. Yin, G. Li, K.-d. Huang, Scalable mapreduce framework on fpga accelerated commodity hardware, in: S. Andreev, S. Balandin, Y. Koucheryavy (Eds.), Internet of Things, Smart Spaces, and Next Generation Networking, in: Lecture Notes in Computer Science, Vol. 7469, Springer Berlin Heidelberg, 2012, pp. 280–294.
[49] M. Zaharia, A. Konwinski, A.D. Joseph, R. Katz, I. Stoica, Improving mapreduce performance in heterogeneous environments, in: Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation, OSDI'08, 2008, Berkeley, CA, USA, pp. 29–42.

Katayoun Neshatpour is a Ph.D. 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. She has received her M.Sc. degree in Electrical Engineering from Sharif University of Technology and B.Sc. degree from Isfahan University of Technology. Her research interests are acceleration of Big data applications with a focus on MapReduce platform, energy-efficient implementation of machine-learning application including Convolutional Neural Networks and low-power VLSI design.

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.



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 Ph.D. 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 M.S. degree from the Department of Information and Computer Science at the UC Irvine in 2010.

Tinoush Mohsenin is an Assistant Professor in the Department of Computer Science and Electrical Engineering at University of Maryland Baltimore County. She received her Ph.D. from University of California, Davis in 2010 and M.S. degree from Rice University in 2004, both in Electrical and Computer Engineering. Prof. Mohsenin's research focus is on designing highly accurate and energy efficient embedded processors for machine learning, signal processing and knowledge extraction techniques for autonomous systems, wearable smart health monitoring, and embedded big data computing. She has over 80 peer-reviewed journal and conference publications and is the recipient of NSF CAREER award in 2017, the best paper award in the GLSVLSI conference in 2016, and the best paper honorable award in ISCAS 2017 for developing domain-specific accelerators for biomedical, deep learning and cognitive computing. She currently leads 8 research projects in her lab which are all funded by National Science Foundation (NSF), Army Research Lab (ARL), Northrop Grumman, Boeing, Nvidia and Xilinx. She has served as associate editor in IEEE Transactions on Circuits and Systems-I (TCAS-I) and IEEE Transactions on Biomedical Circuits and Systems (TBioCAS). She was the local arrangement co-chair for the 50th IEEE International Symposium on Circuits and Systems (ISCAS) in Baltimore. She has also served as technical program committee member of the IEEE International Solid-State Circuits Conference Student Research (ISSCC-SRP), IEEE Biomedical Circuits and Systems (BioCAS), IEEE International Symposium on Circuits and Systems (ISCAS), ACM Great Lakes Symposium on VLSI (GLSVLSI and IEEE International Symposium on Quality Electronic Design (ISQED) conferences. She also serves as secretary of IEEE P1890 on Error Correction Coding for Non-Volatile Memories.

Hassan Ghasemzadeh received the B.Sc. degree from Sharif University of Technology, Tehran, Iran, the M.Sc. from University of Tehran, Tehran, Iran, and the Ph.D. from the University of Texas at Dallas, Richardson, TX, in 1998, 2001, and 2010 respectively, all in Computer Engineering. He was on the faculty of Azad University from 2003 to 2006 where he served as Founding Chair of Computer Science and Engineering Department at Damavand branch, Tehran, Iran. He spent the academic year 2010–2011 as a Postdoctoral Fellow at the West Wireless Health Institute, La Jolla, CA. He was a Research Manager at the UCLA Wireless Health Institute 2011–2013. Currently, he is Assistant Professor of Computer Science in the School of Electrical Engineering and Computer Science at Washington State University, Pullman, WA. The focus of his research is on algorithm design and system level optimization of embedded and pervasive systems with applications in healthcare and wellness.

Houman Homayoun received the B.S. degree in electrical engineering from Sharif University of Technology, Tehran, Iran, in 2003, the M.S. degree in computer engineering from the University of Victoria, Victoria, BC, Canada, in 2005, and the Ph.D. degree from the Department of Computer Science, University of California at Irvine, Irvine, CA, USA, in 2010. He was an NSF Computing Innovation Fellow at the University of California at San Diego, San Diego, CA, USA, for two years. He is currently an Assistant Professor at the Electronics and Communication Engineering Department, George Mason University, Fairfax, VA, USA. He also holds a joint appointment with the Computer Science Department. He is currently leading a number of research projects, including the design of heterogeneous architectures for big data and nonvolatile logics to enhance design security, which are funded by the National Science Foundation, General Motors Company, and the Defense Advanced Research Projects Agency. Dr. Homayoun was a recipient of the NSF Computing Innovation Fellowship from the CRA and CCC.