# Energy-Efficient Acceleration of Big Data Analytics Applications Using FPGAs

Katayoun Neshatpour[1], Maria Malik[1], Mohammad Ali Ghodrat[2], Avesta Sasan[1], and Houman Homayoun[1]

[1]Department of Electrical and Computer Engineering, George Mason University
{kneshatp, mmalik9, hhomayou}@gmu.edu
[2]Computer Science Department, University of California Los Angeles
ghodrat@gmail.com

***Abstract*— A recent trend for big data analytics is to provide heterogeneous architectures to allow support for hardware specialization. Considering the time dedicated to create such hardware implementations, an analysis that estimates how much benefit we gain in terms of speed and energy efficiency, through offloading various functions to hardware would be necessary. This work analyzes data mining and machine learning algorithms, which are utilized extensively in big data applications in a heterogeneous CPU+FPGA platform. We select and offload the computational intensive kernels to the hardware accelerator to achieve the highest speed-up and best energy-efficiency. We use the latest Xilinx Zynq boards for implementation and result analysis. We also perform a first order comprehensive analysis of communication and computation overheads to understand how the speedup of each application contributes to its overall execution in an end-to-end Hadoop MapReduce environment. Moreover, we study how other system parameters such as the choice of CPU (big vs little) and the number of mapper slots affect the performance and power-efficiency benefits of hardware acceleration. The results show that a kernel speedup of upto $\times 321.5$ with hardware+software co-design can be achieved. This results in $\times 2.72$ speedup, $2.13\times$ power reduction, and $15.21\times$ energy efficiency improvement (EDP) in an end-to-end Hadoop MapReduce environment.**

***Index Terms*— machine learning, hardware+software co-design, Zynq boards, MapReduce, Hadoop, FPGA**

## I. Introduction

Advances in various branches of technology - data sensing, data communication, data computation, and data storage - are driving an era of unprecedented innovation for information retrieval. Emerging big data analytics applications require a significant amount of server computational power [1], [2]. However, while demand for data center computational resources continues to grow, the semiconductor industry has reached its physical scaling limits and is no longer able to reduce power consumption in new chips [3].

As chips are hitting power limits, computing systems are moving away from general-purpose designs and toward greater specialization. Hardware acceleration through specialization has received renewed interest in recent years, mainly due to the dark silicon challenge.

Moreover, various types of parallelism have been explored as an alternative to technology scaling to meet the demands of the high performance systems. MapReduce [4], a framework utilized extensively for big-data application, is a well-utilized implementation for processing and generating large data sets, in which the programs are parallelized and executed on a large cluster of commodity servers.

To address the computing requirements of big data, and based on the benchmarking and characterization results, we envision a data-driven heterogeneous architecture for next generation big data server platforms that leverage the power of field programmable gate array (FPGA) to build custom accelerators. We leverage the latest developments in heterogeneous prototype platforms, such as Xilinx Zynq that enables integration of asymmetric cores with FPGA accelerators to build the foundation of server architecture to address performance and energy efficiency requirements of the emerging big data server architecture.

Most recent works focus on the implementation of an entire particular machine learning application or offloading complete phases of MapReduce to the FPGA hardware [5], [6], [7], [8]. 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 the algorithm trades some speedup at a benefit of less hardware.

We study various data mining and machine learning algorithms, namely K-means clustering, K nearest neighbors (KNN), support vector machine (SVM) and Naive Bayes classifier, and profile them in order to find the CPU-intensive and time-consuming kernels (hotspot functions) to offload to the FPGA. Still, the rest of the application, Hadoop file system management, compression and decompression, shuffling and sorting, and standard Java library accesses are performed at GHz clock frequency CPU server. This methodology raises several new challenges and trade-offs in performance, energy, application mapping, HW+SW partitioning, scheduling, and resource allocation.

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 func-

978-1-4799-9926-2/15/$31.00 ©2015 IEEE

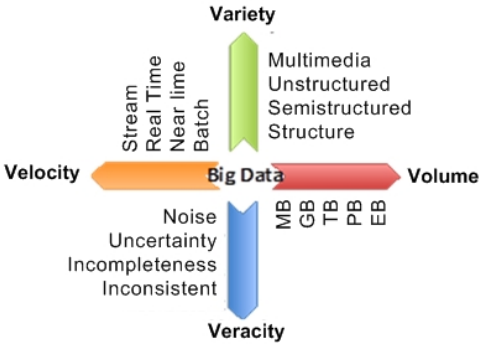


Fig. 1. Illustration of Four Vs of big data.

tions 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 the micro-architecture diversity of the server; big Xeon cores vs little Atom cores.

This paper is organized as follows. In Section II, a background is provided on big data, MapReduce, and Apache Hadoop. In Section III, the end-to-end system architecture utilized for implementation and analysis purposes is introduced. In Section IV, the acceleration of studied benchmarks is described. Section V introduces the analytical assumptions utilized for evaluation of potential speedup and energy-efficiency gain. In Section VII, the results of profiling and hardware implementation of micro kernels and functions within the studied benchmarks are introduced. Section VI introduces Zynq as the case study for HW+SW acceleration. Section IX discusses the related work and Section X sums up the conclusions.

## II. 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 are 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 [9]. 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

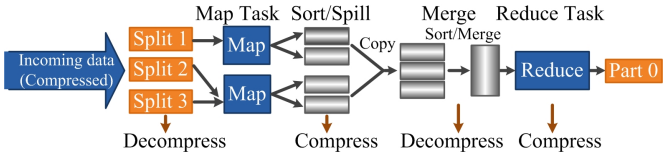


Fig. 2. Hadoop MapReduce: Computational Framework Phases [11].

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 64MB, 128MB, etc.) [10].

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.

## III. System Architecture

The system studied in this paper, consists of a high-performance CPU as the master node, which is connected to several Zynq devices as slave nodes.

The master node runs the HDFS, and is responsible for the job scheduling between all the slave nodes. It is configured to distribute the computation workloads among the slave nodes (worker nodes), as shown in Fig. 3. Each worker node has a fixed number of map and reduce slots, the number of which, is statistically configured.

Is this paper, we compare two types of server platform as the master node; Intel Atom C2758, and Intel Xeon E5. These two types of servers represent two schools of thought on server architecture design: using big Xeon cores, 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 [12], [13], [14].

Intel Atom C2758 server deploys 8 processor cores per node, a two-level cache hierarchy (L1 and L2 cache sizes of 24KB and 1024KB, respectively), and an operating frequency of 2.4GHz with Turbo Boost. Intel Xeon E5-2420 has two socket of six aggressive processor cores per node, three levels of cache hierarchy: L1 and L2 cache are private to each core

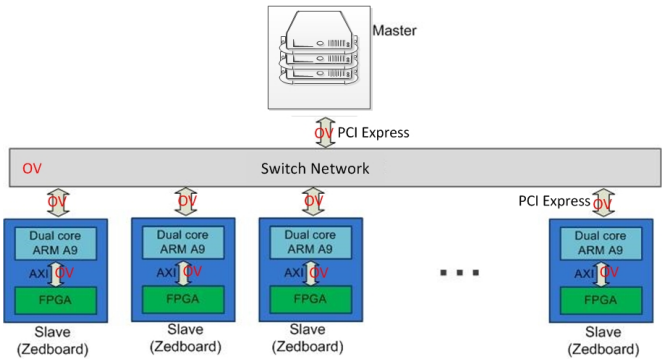


Fig. 3. System architecture.

while LLC (L3) is shared among all cores. L1, L2 and L3 cache sizes are 32KB, 256KB and 15MB, respectively.

In the system architecture, each slave node is equipped with a Zynq device, which is considered as a mapper/reducer slot. The Zynq devices are ZedBoards featuring XC7Z020 Zynq SoCs. The ZedBoard integrates two 667 MHz ARM Cortex-A9 with an Artix-7 FPGA with 85 KB logic cells and 560 KB block RAM.

Thus, ZedBoard is divided into two partitions - the ARM Cortex-A9 processor-based processing system (PS) and the FPGA being the programmable logic (PL). The connections between the PL and PS is established through the AXI interconnect, which will be described in more detail in Section VI.

The basic idea for designing the switching network in Fig. 3, is to have a network with enough capacity at a reasonable cost, so that all components can communicate with each other at an acceptable speed. The master and slave nodes communicate with each other through the switching network, which is implemented with the PCI-Express.

In order to have a better estimation of the speedup gains in the architecture in Fig. 3, various overheads need to be taken into account, which include the overhead of the data transfer time between the nodes in the network through the PCI-Express, the overhead of the switching network, and the data transfer time between the ARM core (PS) and the FPGA (PL) in the ZedBoard. These overheads have been marked with "OV" in Fig. 3.

## IV. 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 [15].

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. We select four widely used application for HW+SW co-design. We characterize each application in order to find out which kernels are the best candidates to be offloaded to the hardware for acceleration.

TABLE I
Profiling results.

| Application | Function | calls | percentage | time/call |
|---|---|---|---|---|
| K-means | Example 1 | $N = 100$ | $k = 13$ | $f = 10$ |
| | K_means_clustering | 80 | 100% | $500\mu s$ |
| | find_nearest_point | 10321 | 75% | $2.91\mu s$ |
| | euclid_dist_2 | 89783 | 50% | $223ns$ |
| | Example 2 | $N = 17695$ | $k = 13$ | $f = 18$ |
| | K_means_clustering | 80 | 99.9% | $358ms$ |
| | find_nearest_point | 17025982 | 84.6% | $1.424\mu s$ |
| | euclid_dist_2 | 134110063 | 75.7% | $162ns$ |
| KNN | Example1 | | f=5 | |
| | KNN_classify | 18597 | 97.07% | $889.29\mu s$ |
| | sq_euclid_dist | 280004358 | 80.26% | $59.39ns$ |
| | Example2 | | f=10 | |
| | KNN_classify | 18597 | 99% | $1.571ms$ |
| | sq_euclid_dist | 280004358 | 88.28% | $114.92ns$ |
| SVM | Example 1 | $N = 610$ | $M = 370$ | $f = 9930$ |
| | solve_dual | 7136 | 78.3% | $375.56\mu s$ |
| | sprod_ns | 2667530 | $8.2\%s$ | $105ns$ |
| | sprod_ss | 245091 | 2.6% | $367ns$ |
| | Example 2 | $N = 2000$ | $M = 879$ | $f = 9947$ |
| | solve_dual | 7136 | 91.41% | $388.17\mu s$ |
| | spro_ns | 2667530 | $8.5\%s$ | $97.5ns$ |
| | sprod_ss | 245091 | 1.3% | $163ns$ |
| Naive Bayes | linelength | 18597 | 13.85% | $2.094\mu s$ |
| | areaunderCurve | 18597 | 13.48% | $1.997\mu s$ |
| | normDecay | 18597 | 11.61% | $1.687\mu s$ |
| | abs | 9521459 | 11.98% | $3.3ns$ |
| | nb_classify | 18597 | 8.99% | $1.205\mu s$ |

### A. 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 I shows the profiling results for selected examples for each application. A detailed description of the results for each application comes in the sequel.

*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 [16] 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 behaviour 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 I, 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%.

*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 I 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.

*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 [17] for different data sets. *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 I, the number of training documents ($N$), support vectors ($M$) and feature size ($f$) are important factors that influence the execution time.

*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 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.

### B. 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 [18], [19]. 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 [20]. Recent works have addressed the problem of the implementation of dynamic, pointer-based structures [18], [19]. 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 II
HLS IMPLEMENTATION RESULTS

| Application | Function | clock[ns] | latency[cycles] | interval[cycles] |
|---|---|---|---|---|
| K-means $N = 10, k = 13, f = 10$ | K_means_clustering | 4.01 | 686 | 687 |
| | find_nearest_point | 3.36 | 2 | 3 |
| | euclid_dis_2 | 3.36 | 1 | 2 |
| K-means $N = 17695, k = 13, f = 18$ | K_means_clustering | 4.01 | 18697 | 18698 |
| | find_nearest_point | 3.38 | 2 | 3 |
| | euclid_dis_2 | 3.38 | 1 | 2 |
| KNN classify $f = 5$ | sq_euclid_dist | 5.2 | 154 | 3 |
| KNN classify $f = 10$ | sq_euclid_dist | 5.2 | 224 | 5 |
| SVM learn $N = 610, M = 370, f = 9947$ | solve_dual | 4.55 | 105 | 106 |
| | sprod_ns | 3.65 | 3210 | 100 |
| | sprod_ss | 3.65 | 5113 | 150 |
| SVM learn $N = 2000, M = 879$ $f = 9930$ | solve_dual | 4.55 | 3327 | 3328 |
| | sprod_ns | 3.65 | 3210 | 100 |
| | sprod_ss | 3.64 | 5113 | 150 |
| Naive Bayes | lineLength | 3.65 | 4116 | 128 |
| | areaUnderCurve | 3.65 | 4134 | 128 |
| | normDecay | 4.13 | 4146 | 128 |
| | abs | 3.13 | 4 | 1 |
| | nb_classify | 4.55 | 957 | 25 |

Table II 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 can start to accept new input data, which is an indicator of the speedup the function can achieve.

## V. HARDWARE+SOFTWARE SPEEDUP CALCULATION

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.

### A. Zero-overhead communication

In Section IV 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.

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

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 $i$, respectively and $C_i$ is the the number of calls to function $i$.

### B. Modeling the overhead

The assumptions for the calculation of the overhead due to PS-PL and the PCI-Express 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.

*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.

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

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:

$$T_{i,tr} = \frac{D_i}{BW_{PL,PS}}, \quad (3)$$

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.

*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.

## VI. 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 100MHz 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 400MBps and

300MBps, respectively, which are 99.76% and 74.64% of the theoretical bandwidths [21]. These numbers are utilized to find the data transfer overhead between the PL and PS in the Zynq devices.

## VII. Implementation Results

### A. Acceleration results on the Zedboard

Table III 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 III 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 III.

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 III, 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 III 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.

TABLE III
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 |
| | $N = 17695, k = 13, f = 18$ | | |
| | 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 |

### B. 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 [22]. Using Vtune, we analyze the contribution of kernel execution time over the total execution time when running Hadoop.

Fig. 4 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 III and Hadoop hotspot analysis in Fig. 4.

Fig. 5 shows the achievable acceleration of the Hadoop system for selected examples in Table III 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 $312\times$ with zero overhead, the speedup drops to $146\times$ with the data transfer overhead, and $2.72\times$ and $2.78\times$ 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 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.

### C. 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

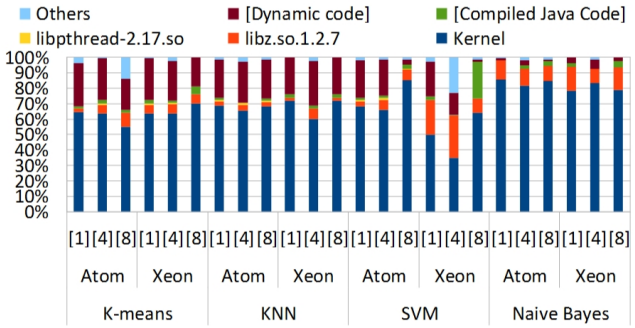


Fig. 4. Hotspot analysis before acceleration.

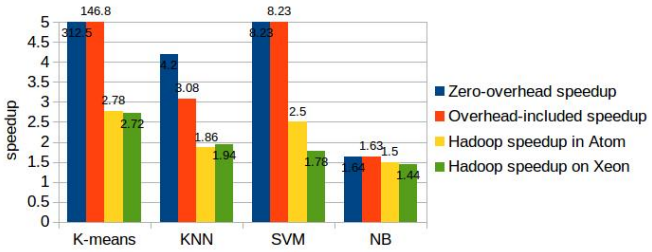


Fig. 5. Acceleration on a full-blown system with four mapper slots.

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 V. 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 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 IV 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 IV, power is reduced by up to 3.7× in the

TABLE IV

POWER AND ENERGY ANALYSIS FOR FOUR MAPPER SLOTS

| Application | ini power[$w$] | acc power[$w$] | ini EDP[$ws^2$] | acc EDP[$ws^2$] | 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 |

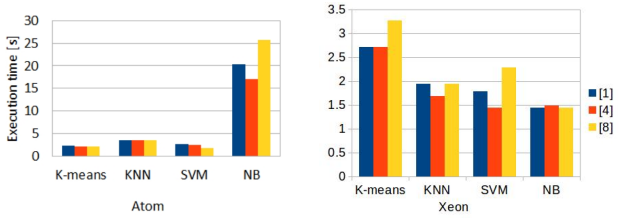


Fig. 6. Execution time for various number of mapper slots on Atom and Xeon

studied applications. Moreover, since both the execution time and the power has been reduced through the acceleration, EDP is significantly reduced by upto 15.21×.

## VIII. SENSITIVITY ANALYSIS

### A. 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. 4 shows the results of the hotspot analysis for different number of mappers. Fig. 6 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. 6 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.

TABLE V
EXECUTION TIME FOR DIFFERENT DATA SIZES

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

### B. Size of data

We have conducted the data size sensitivity analysis of studied Hadoop machine learning applications on the Xeon with 4 mappers to understand the impact of size of data on the speedup. Table V shows the execution time after acceleration and the speedup for different data sizes for all applications. Table V shows that as expected, the execution time increases with the increase in data size. Moreover, the data size has little impact on the the achievable speedup, which is due to the overhead for the transfer of data through the switching network and PCI-express.

### C. Big and Little cores

In exploring the choice of architecture for big data, we compare the performance of big data applications on two very distinct micro-architectures; 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. 5, the range of speedup achieved through HW+SW co-design is the almost similar for Atom and Xeon for the K-means, KNN, and Naive Bayes. For the SVM, Atom yield a higher speedup.

However, we are more interested in the overall execution time. Fig 6 shows that the overall execution time is lower on the the architecture in which, HDFS runs on the high-end cores (Xeon), especially for the K-means and the Naive Bayes algorithm.

Based on Table IV, the Atom server yields lower power and better energy efficiency both before and after the acceleration. 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. Moreover, the energy efficiency is enhanced more significantly for the high-end server (Xeon). Overall, the results indicate the benefit of FPGA acceleration in both high-end and low-end server platforms.

## IX. RELATED WORK

In [8], FPMR is introduced as a MapReduce framework on the FPGA with RankBoost as a case study, which adopts a dynamic scheduling policy for better resource utilization and load balancing. In [7], a hardware accelerated MapReduce architecture is implemented on Tilera's many core processor board. In this architecture data mapping, data merging and data reducing processing are offloaded to the accelerators. The Terasort benchmark is utilized to evaluate the proposed architecture. In [6] hardware acceleration is explored through an eight-salve Zynq-based MapReduce architecture. It is implemented for a standard FIR filter to show the benefits gained through hardware acceleration in the MapReduce framework, where the whole low-pass filter is implemented on the FPGA. In [23], a configurable hardware accelerator is used to speed up the processing of multi-core and cloud computing applications on the MapReduce framework. The accelerator is utilized to carry out the reduce tasks.

In [5], a detailed MapReduce implementation of the K-means application is done in the HW+SW framework, which enhances the speedup of a non-parallel software implementation of K-means. While the Hadoop implementation of the application explores their inherent parallelism and enhances their performance with respect to non-MapReduce software implementation, in this paper, we aim to model the range of potential speedup that can be achieved through HW+SW co-design and compare the resulting speedup to the MapReduce implementation. In a recent work [24], [25], we provide an estimation of hardware acceleration speedup using hardware/software co-design of machine learning and data mining applications on CPU+FPGA platform. However the communication overhead was not studied thoroughly in our previous work. In addition this work is different as it also studies system level parameters of hadoop MapReduce framework as well as architecture level parameters on the HW+SW acceleration potentials.

## X. CONCLUSIONS

To significantly improve performance and energy-efficiency of processing big data analytics applications, in this paper a heterogeneous architecture that integrates general-purpose CPUs with dedicated FPGA accelerators was studied. Full Hadoop MapReduce profiling was used to find the hot regions of several widely used machine learning and data mining applications. The hardware equivalents of performance hot regions were developed using high level synthesis tools. With HW+SW co-design, we offloaded the hot regions to the hardware to find the design with the highest speed-up. A comprehensive analysis of communication and computation overheads in Hadoop was performed to understand how the speedup of each application contributes to its overall execution in an end-to-end Hadoop MapReduce environment. Sensitivity analysis was performed on parameters such as the number of mapper slots and CPU micro-architecture. The results show that a kernel speedup of upto $\times 321.5$ with HW+SW co-design can be achieved. This results in $\times 2.72$ speedup, $2.13\times$ power reduction and $15.21\times$ energy-efficiency improvement (EDP) in an end-to-end Hadoop MapReduce environment considering various data transfer and communication overheads.

## REFERENCES

[1] M. Malik and H. Homayoun, "Big data on low power cores are low power embedded processors a good fit for the big data workloads," *33rd IEEE International Conference on Computer Design*, 2015.

[2] M. Malik, S. Rafatirah, A. Sasan, and H. Homayoun, "System and architecture level characterization of big data applications on big and little core server architectures," *IEEE International Conference on Big Data. IEEE BigData*, 2015.

[3] V. Kontorinis, L. Zhang, B. Aksanli, J. Sampson, H. Homayoun, E. Pettis, D. Tullsen, and T. Simunic Rosing, "Managing distributed ups energy for effective power capping in data centers," in *Computer Architecture (ISCA), 2012 39th Annual International Symposium on*, June 2012, pp. 488–499.

[4] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," in *OSDI 2004*, 2004.

[5] Y.-M. Choi and H.-H. So, "Map-reduce processing of k-means algorithm with fpga-accelerated computer cluster," in *IEEE ASAP*, June 2014, pp. 9–16.

[6] Z. Lin and P. Chow, "Zcluster: A zynq-based hadoop cluster," in *2013 FTP*, Dec 2013, pp. 450–453.

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

[8] Y. e. a. Shan, "Fpmr: Mapreduce framework on fpga," in *Proc. ACM/SIGDA Int Symp FPGA*, 2010, pp. 93–102.

[9] "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.

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

[11] "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.

[12] N. e. Hardavellas, "Toward dark silicon in servers," *IEEE Micro*, vol. 31, pp. 6–15, 2011.

[13] H. Homayoun, V. Kontorinis, A. Shayan, T.-W. Lin, and D. Tullsen, "Dynamically heterogeneous cores through 3d resource pooling," in *High Performance Computer Architecture (HPCA), 2012 IEEE 18th International Symposium on*, Feb 2012, pp. 1–12.

[14] V. Kontorinis, M. K. Tavana, M. H. Hajkazemi, D. M. Tullsen, and H. Homayoun, "Enabling dynamic heterogeneity through core-on-core stacking," *The 51st Annual Design Automation Conference 2014, DAC '14, San Francisco, CA, USA*, 2014.

[15] e. a. Van Craeynest, "Scheduling heterogeneous multi-cores through performance impact estimation (pie)," in *ISCA 2012*, 2012, pp. 213–224.

[16] e. a. Narayanan, "Minebench: A benchmark suite for data mining workloads," in *IEEE Int Symp Workload Characterization*, Oct 2006, pp. 182–188.

[17] T. Joachims, "Making large-scale SVM learning practical," in *Advances in Kernel Methods - Support Vector Learning*, B. Schölkopf, C. Burges, and A. Smola, Eds. Cambridge, MA: MIT Press, 1999, ch. 11, pp. 169–184.

[18] F. Winterstein, S. Bayliss, and G. Constantinides, "High-level synthesis of dynamic data structures: A case study using vivado hls," in *2013 FTP*, Dec 2013, pp. 362–365.

[19] L. Smria, K. Sato, and G. D. Micheli, "Synthesis of hardware models in c with pointers and complex data structures," *IEEE TVLSI*, vol. 9, no. 6, pp. 743–756, 2001.

[20] "Vivado design suite user guide: High-level synthesis," http://www.xilinx.com/support/documentation/sw_manuals/xilinx2013_2/ug902-vivado-high-level-synthesis.pdf.

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

[22] "Intel vtune amplifier xe performance profiler." http://software.intel.com/en-us/articles/intel-vtune-amplifier-xe/, accessed: 2014-11-30.

[23] C. Kachris, G. C. Sirakoulis, and D. Soudris, "A configurable mapreduce accelerator for multi-core fpgas (abstract only)," in *Proc ACM/SIGDA Intl Symp FPGAs*, 2014, pp. 241–241.

[24] K. Neshatpour, M. Malik, M. A. Ghodrat, and H. Homayoun, "Accelerating big data analytics using fpgas," *23rd IEEE Annual International Symposium on Field-Programmable Custom Computing Machines, FCCM*, p. 164, 2015.

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