2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing

Accelerating Machine-Learning Kernels in Hadoop Using FPGAs

Katayoun Neshatpour[1], Maria Malik[1], and Houman Homayoun[1]

[1]Department of Electrical and Computer Engineering, George Mason University

***Abstract*— Big data applications share inherent characteristics that are fundamentally different from traditional desktop CPU, parallel and web service applications. They rely on deep machine learning and data mining applications. A recent trend for big data analytics is to provide heterogeneous architectures to allow support for hardware specialization to construct the right processing engine for analytics applications. However, these specialized heterogeneous architectures require extensive exploration of design aspects to find the optimal architecture in terms of performance and cost. This paper analyzes how offloading computational intensive kernels of machine learning algorithms to a heterogeneous CPU+FPGA platform enhances the performance. We use the latest Xilinx Zynq boards for implementation and result analysis. Furthermore, we perform a comprehensive analysis of communication and computation overheads such as data I/O movements, and calling several standard libraries that can not be offloaded to the accelerator to understand how the speedup of each application will contribute to its overall execution in an end-to-end Hadoop MapReduce environment.**

***Index Terms*— Big data, acceleration, FPGA**

I. Introduction

Emerging big data analytics applications require a significant amount of server computational power. Big data analytics applications heavily rely on big-data-specific deep machine learning and data mining algorithms, and are running complex database software stack. This set of characteristics is necessitating a change in the direction of server-class microarchitecture to improve their computational and memory efficiency.

MapReduce [1] is the programming model developed by Google to handle large-scale data analysis. The MapReduce framework is extensively utilized for big-data application and it 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. MapReduce consists of map and reduce functions where, the map functions parcel out work to different nodes in the distributed cluster, and the reduce functions collate the work and resolve the results. The map functions 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.

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 includes numerous micro-benchmarks including clustering, classifiers and data compression benchmarks. The Hadoop MapReduce implementation consist of several phases as depicted in Fig. 1.

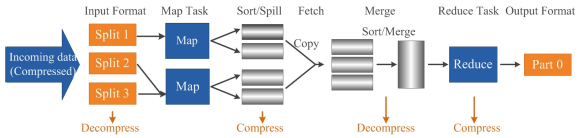

Fig. 1. Hadoop MapReduce: Computational Framework Phases [2].

Hardware acceleration through specialization has received renewed interest in recent years, mainly due to the dark silicon challenge. Heterogeneous architectures comprise different types of cores, domain-specific accelerators, as well as programmable fabrics (FPGA) to synthesize a custom accelerator. Tight integration between the general-purpose processor and the programmable logic can provide enormous opportunities to add any type of custom-designed accelerator. An example of such architecture is ZYNQ (ARM+FPGA).

To address the computing requirements of big data, and based on the benchmarking and characterization results, we envision a heterogeneous architecture for next big data server platforms that leverage the power of FPGA to build custom accelerators. Our baseline architecture consists of a standard Hadoop platform which is extended with FPGA hardware for acceleration. To evaluate how hardware acceleration can address performance demands of big data analytics applications, we study various data mining and machine learning algorithms to find the CPU-intensive and time consuming kernels (hotspot functions) to offload to FPGA. We assume the applications are fully known, therefore we can find the best possible application-to-core+FPGA match. We calculate the acceleration gained through the hardware-software co-design process and evaluate how it will speedup the Hadoop end-to-end system.

The paper is organized as follows. In Section II the related work is discussed. Section III describes the end-to-end system architecture. Section IV and V describe the methodology for modeling the kernel speedup and overall Hadoop speedup, respectively. Section VI reports the implementation results. Section VII discusses effects of design parameters. Finally, Section VIII concludes the paper.

II. Related work

In [3] 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

978-1-4799-8006-2/15 $31.00 © 2015 IEEE
DOI 10.1109/CCGrid.2015.165

to enhance the load balancing. In [4], a hardware accelerated MapReduce architecture is implemented on Tilera's manycore platform. 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 [5] 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 entire low-pass filter is implemented on the FPGA.

In [6], a detailed MapReduce implementation of the K-means application is done in a hardware-software framework, which enhances the speedup of a non-parallel software implementation of K-means. While the Hadoop implementation of the application explores their inherent data 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 hardware-software co-design and compare the resulting speedup to the MapReduce implementation.

Most recent works focus on the implementation of an entire particular machine learning application or offloading complete phases of MapReduce to the FPGA. However, the implementation of the entire algorithm on FPGA would result in excessive hardware and requires significant design effort. Thus, this paper focuses on hardware-software co-design of the algorithm that will trade some speedup at a benefit of less hardware and higher programmability.

III. System Model

Hadoop runs the job by breaking it into tasks, i.e., map tasks and reduce tasks. 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 running.

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 and running the HDFS among other tasks. The master node is configured to distribute the computation workloads among the slave nodes (worker nodes), as shown in Fig. 2. Each worker node has a fixed number of map and reduce slots, the number of which, is statistically configured. (In this paper one mapper slot per each Zynq device is assumed)

We used Intel Atom C2758 as the master node; Atom server represents a new trajectory in server design that advocates the use of a low-power core to address the dark silicon challenge facing servers [7]. Intel Atom C2758 server has 8 processors, 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.

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

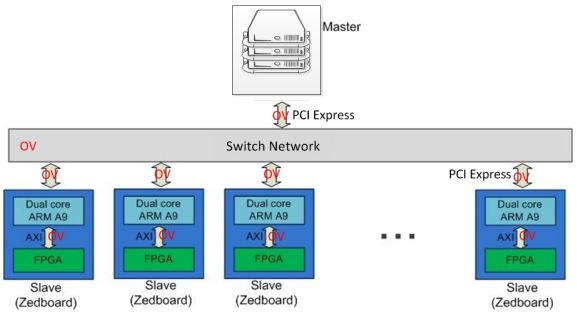

Fig. 2. System architecture.

Thus, the 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.

The basic idea for designing the switching network in Fig. 2, 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. For this reason, the PCI-Express is being deployed for the interconnection. The master and slaves nodes communicate with each other through the PCI-Express.

In order to have a better estimation of the speedup gains in the architecture in Fig. 2, various overheads need to be taken into account, which include the overhead of the data transfer 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. 2.

IV. Kernel Acceleration through Hardware-Software co-design

The primary step to estimate the benefit obtained through hardware-software co-design in an end-to-end MapReduce platform, is to carry out a comprehensive workload analysis and performance monitoring for individual applications. To this end, we selected four widely used applications, namely K-means clustering, KNN classification, SVM-learn and Naive Bayes classification, and characterized them in order to find their hotspot functions to be offloaded to the hardware.

K-means clustering is a partitioning based clustering application that partitions n observations into K clusters such that each observation is mapped to the cluster with the closet mean based on specific features. KNN classification is a pattern recognition algorithm, which finds the K nearest neighbors of a vector among N training vectors. SVM is a machine learning algorithm that is used extensively in data analysis and pattern recognition as non-probabilistic binary linear classifier. Naive Bayes classifier is another machine learning algorithm which is used as a probabilistic classifier using strong independent feature model and the Bayesian theorem.

These applications were profiled using the GNU profiler. Functions which accounted for significant part of the the execution time were selected for hardware acceleration. Xilinx Vivado HLS tool was utilized to create register transfer

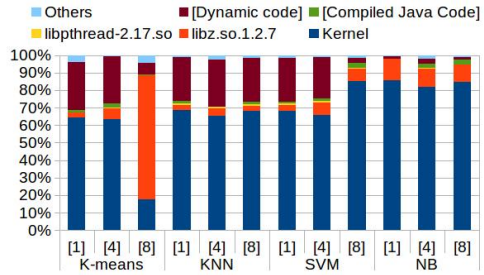

Fig. 3. Hotspot analysis before acceleration.

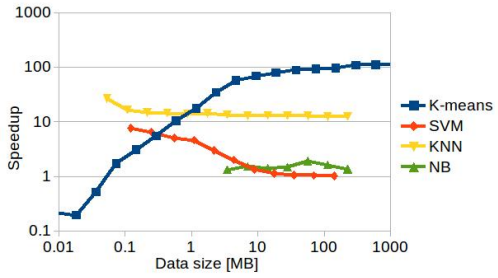

Fig. 4. kernel acceleration for different input data sizes.

level (RTL) equivalents for the hotspot functions. High-level synthesis is the automated process of transforming a C or C++ code into an RTL design.

The speedup gained through offloading hotspot functions to the hardware was calculated according to (1).

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

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 the function i, respectively, C_i is the the number of calls to function i, D_i is the size of data being transferred between the PL and PS during each call of the accelerated function and $BW_{PL,PS}$ is the bandwidth of the data transfer between the PL and PS (i.e. BW of the AXI for the Zedboard).

A. Sensitivity of Kernel Acceleration to Input Data Size

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.

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 based on (1).

V. Hadoop Acceleration

In Section IV, we elaborated how a specific algorithm is accelerated on the Zynq platform. However, not all the execution time of a MapReduce implementation of an algorithm is spent on the kernel. Data compression/decompression, calling several standard libraries, data I/O movements, etc., will take up a considerable part of the execution time. Thus, the speedup derived in Section IV will only contribute to the fraction of the execution time that corresponds to the kernel. Based on the fraction of time each kernel execution contributes to the Hadoop end-to-end system, the overall speedup is derived using Admahl's law.

We use Intel Vtune for hotspot analysis to find the regions of Hadoop MapReduce that remains on the CPU. Intel VTune is a performance-profiling tool that provides an interface to the processor performance counters [8]. Using Vtune, we analyze the contribution of kernel execution time over the total execution time when running Hadoop MapReduce.

Fig. 3 shows the common hotspot modules of big data applications for 1, 4 and 8 number of mapper slots. The input split size was set to 64MB in the simulations. In Fig. 3, application kernel represents the computation part to perform the tasks such as K-means, KNN, etc., Libz performs the data compression and decompression tasks for the Hadoop workload. The numbers in the brackets represent the number of mapper slots in the the Hadoop platform.

VI. Implementation Results

A. Results of kernel Speedup Sensitivity to Data Size

As discussed in Section IV, different data input sizes result in different profiling behaviors and thus different speedups. Fig. 4 shows the speedup results for a wide 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 0.1s), the profiling results were not reliable and thus not reported.

As results shown, 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 10MB for these applications), the speedup values start to converge.

B. Hadoop Acceleration Results

Fig. 5 shows the speedup on the Zynq platform and a fully developed Hadoop with Atom as the master node, 4 number of mapper slots and input splits of 64MB. The figure shows how the speedup achieved on each application through hardware

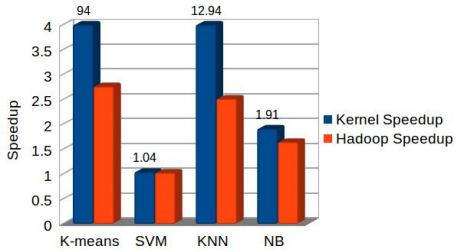

Fig. 5. Acceleration speedup on the Zynq and Hadoop platform.

acceleration is translated into a lower speedup on and end-to-end Hadoop system.

As an example, while the acceleration of the K-means algorithm yields a speedup in the order of $\times 94$, the speedup is reduced to $\times 2.76$ on an end-to-end Hadoop implementation. Thus, the final speedup is greatly affected by the fraction of time the kernel execution takes in the Hadoop environment and the original speedup due to hardware acceleration. In the case of the SVM for instance, since the original kernel speedup was very low (i.e. 4%), the overall speedup is also very low (2.5%).

VII. Design space analysis

Mapping of applications to a heterogeneous architecture to benefit from the diverse core and accelerators is a complex problem, particularly because different phases of the same application will often prefer different cores or configurations and thus, require specific scheduling and mapping to find the best match. Making wrong scheduling decisions can lead to suboptimal performance and negatively impact power and energy consumption as well [9]. Moreover, with big data applications running various softwares like database software stacks, including Hadoop, MapReduce, and message passing interface (MPI), the OS and I/Os are exercised extensively, becoming first-order performance determinants and, therefore, mapping decisions become an even more complex and difficult problem to solve.

The size of input splits, utilizing small or big core, the memory configuration, etc. will impact the performance of the overall architecture. In this section, we explore the potential effect of input splits on the performance of the overall system.

A. Size of Data Splits

In the Hadoop platform, the input data is split into a number of input splits. Each map task processes a logical split of this data that resides on the HDFS. If the input splits are too small, better load balancing can be achieved among the mapper slots, however the overhead on managing the input splits will increase. In order to perform data locality optimization, it is best to run the map task on a node where the input data resides in the HDFS. Thus, the best split size for data would be the size of an HDFS block (64MB) [10].

The results reported in Section VI were based on input splits of 64MB size. Fig. 4 illustrated that other data sizes can yield better speedup for some algorithms. For the SVM algorithm for instance, while data sizes of 64MB yield only 4% increase in the execution time, lower data size can result in higher speedup (i.e. up to $9\times$ speedup for data sizes of 100KB). However, the fraction of time spent in the kernel in an end-to-end Hadoop will be different for smaller input splits. Thus, selection of the input split sizes in an end-to-end system implementation requires an in-depth analysis of sensitivity of both kernel acceleration and the the end-to-end Hadoop execution time to the input split size.

VIII. Conclusion and Future Work

To significantly improve performance of processing big data analytics applications, 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 hot regions were developed using high level synthesis tools. With hardware-software 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 was used to understand how the speedup of each application will contribute to its overall execution in an end-to-end Hadoop MapReduce implementation. Sensitivity analysis was performed on the size of data input splits to understand the speedup sensitivity to size of data. The results show that a kernel speedup of upto $\times 94$ with hardware-software co-design can be achieved. This results in $\times 2.69$ speedup in an end-to-end Hadoop MapReduce environment considering various data transfer and communication overheads.

Future work will focus on detailed implementation of the kernels including the hardware-software co-design of individual map and reduce functions for each application. Comprehensive and experimental exploration of other design aspects, including number of mapper slots, type of master core and hotspot analysis of the Hadoop environment for a wide range of input split sizes will allow the development of a more generalized and accurate model to better understand the performance gains of using FPGA to accelerate big data analytics applications.

References

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

[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] Y. Shan, B. Wang, J. Yan, Y. Wang, N. Xu, and H. Yang, "FPMR: Mapreduce framework on FPGA," in *Proc Annual ACM/SIGDA Int Symp Field Programmable Gate Arrays*, 2010, pp. 93–102.

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

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

[6] Y.-M. Choi and H.-H. So, "Map-reduce processing of k-means algorithm with FPGA-accelerated computer cluster," in *IEEE Int Conf Application-specific Systems, Architectures and Processors*, June 2014, pp. 9–16.

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

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

[9] K. Van Craeynest, A. Jaleel, L. Eeckhout, P. Narvaez, and J. Emer, "Scheduling heterogeneous multi-cores through performance impact estimation (pie)," in *Proc. . 39th Annual Int Symp Computer Architecture (ISCA)*, 2012, pp. 213–224.

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