System and Architecture Level Characterization of Big Data Applications on Big and Little Core Server Architectures

Maria Malik[1], Setareh Rafatirah[2], Avesta Sasan[1], Houman Homayoun[1]
[1]Department of Electrical and Computer Engineering, [2]Department of Information Sciences and Technology
George Mason University, Fairfax, VA, USA
{mmalik9, srafatir, hhomayou}@gmu.edu

Abstract*— Emerging Big Data applications require a significant amount of server computational power. Big data analytics applications rely heavily on specific deep machine learning and data mining algorithms, and exhibit high computational intensity, memory intensity, I/O intensity and control intensity. Big data applications require computing resources that can efficiently scale to manage massive amounts of diverse data. However, the rapid growth in the data yields challenges to process data efficiently using current server architectures such as big Xeon cores. Furthermore, physical design constraints, such as power and density, have become the dominant limiting factor for scaling out servers. Therefore recent work advocates the use of low-power embedded cores in servers such as little Atom to address these challenges. In this work, through methodical investigation of power and performance measurements, and comprehensive system level and micro-architectural analysis, we characterize emerging big data applications on big Xeon and little Atom-based server architecture. The characterization results across a wide range of real-world big data applications and various software stacks demonstrate how the choice of big vs little core-based server for energy-efficiency is significantly influenced by the size of data, performance constraints, and presence of accelerator. Furthermore, the microarchitecture-level analysis highlights where improvement is needed in big and little cores microarchitecture.

***Index Terms*—Performance, Power, Characterization, Big Data, High-Performance server, Low-Power server, Accelerator**

1. 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. The world of Big Data is constantly changing and producing huge amounts of data that creates challenges to process the applications using existing solutions. Big data applications require computing resources and storage subsystems that can scale to manage massive amounts of diverse data. Individuals, businesses, governments, and society as a whole now have access to enormous collections of big data, empowering them to build their own analytics. Datacenters are therefore required to introduce more nodes to their infrastructure or replace their existing hardware with more powerful systems to respond to this growing demand. This trend increases the infrastructure cost and power consumption. We believe this is the right time to identify the right computing platform for Big Data analytics processing that can provide a balance between processing capacity and power efficiency.

Emerging data applications, in particular from web service domain, share many inherent characteristics that are fundamentally different from traditional desktop, parallel, and scale-out applications [2]. Big data analytics applications in these domains heavily rely on big-data-specific deep machine learning and data mining algorithms, and are running complex database software stack with significant interaction with I/O and OS, and exhibit high computational intensity, memory intensity, I/O intensity and control intensity. In addition, unlike conventional CPU applications, big data applications combine a high data rate requirement with high computational power requirement, in particular for real-time and near-time performance constraints.

This new set of characteristics is necessitating 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 its physical 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 factor for scaling out data centers [3, 4, 5, 6]. Current server designs, based on commodity homogeneous processors, will therefore not be the most efficient in terms of performance/watt to process big data applications [6, 7]. In this work we show that while high performance big cores are optimized for traditional CPU applications, for big data they are very inefficient and are not satisfying their computational-efficiency requirements.

In exploring the choice of server architecture for big data, in this paper, we present a comprehensive analysis of the measurement of power and performance of big data applications on two very distinct microarchitectures; a high performance big Xeon core and another 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 [7, 8, 9, 10, 11]. In addition to power and performance study, we have also performed the Energy-Delay^X Product (ED^{X}P) analysis to evaluate the trade-off between power and performance to understand how near real-time performance constraints for big data analytics affects the choice of big vs. little core server as a more efficient architecture. As for big data applications, achieving a higher processing rate of large amount of data is a prime target, so we have evaluated the processing capability under different data size (per node) by using two metrics – Data Processed Per Second (DPS) and Data Processed Per Joule (DPJ). The analysis helps us understand how the choice of big vs. little cores introduces significant tradeoff in performance, power, energy-delay, and

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

processing capacity for efficient and near-time processing of big data applications. The results demonstrate that while in most applications, server with little cores is more efficient in terms of EDP and DPJ, with constraints for near real-time performance the most efficient server architecture depends on the data size and the type of application.

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. To find out the right architecture for big data processing, it is important to understand how deploying an accelerator, such as FPGA, would necessitate adapting the choice of big vs. little cores. The post acceleration code characteristics are important to find the right architecture for efficient processing of big data applications. For this purpose, we analyze the choice of big vs. little core-based servers for the code that remains for the CPU after assuming the hotspots are offloaded to an accelerator, compared with the choice of big vs. little before acceleration.

Overall, our characterization results across a wide range of real-world big data applications and various software stacks demonstrate how the choice of big vs little core-based servers for energy-efficiency is significantly influenced by the size of data, performance constraints, and presence of accelerator.

To provide insight into whether current design of big and little core requires improvement in the microarchitecture parameters for efficient big data processing we further perform a comprehensive microarchitecture characterization. This study assists in determining whether big data workloads require innovation in microprocessor microarchitecture design.

Contributions: This paper makes the following key contributions:

- *We analyze the measurements of performance and power of Big Data applications on two state-of-the-art server platforms, one with Intel™ Xeon; Big cores and the other with Intel™ Atom; Little cores. We compare the results with traditional CPU, parallel and scaleout applications.*
- *We analyze the performance of hotspot tasks involve in an end-to-end big data analytics running various software stacks including Hadoop MapReduce and Apache Mahout. This includes read and write to HDFS, sorting, compression and decompression, mapping and reduction and calling several standard libraries.*
- *We demonstrate how the size of data, type of application (e.g. I/O intensive and compute Intensive), and performance constraints affect the choice of big vs little core-based servers for efficient big data processing.*
- *We show how offloading map and reduce tasks to an accelerator such as FPGA affects the choice of big vs. little core-based servers for the remaining code on the CPU.*
- *We perform a comprehensive microarchitecture analysis to find the performance bottlenecks in both big and little cores when running big data applications.*

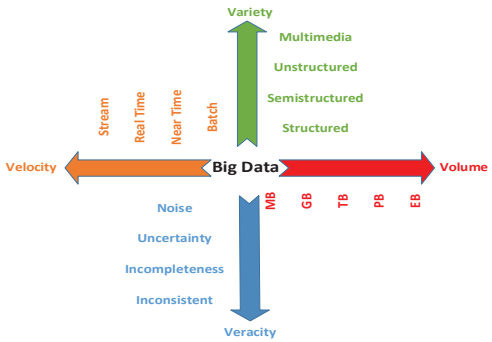


Figure 1: Illustration of Four "Vs" of Big Data

The rest of the paper is organized as follows. Section 2 provides background for big data. Section 3 describes the studied big data, scale-out and Traditional serial and parallel CPU benchmarks. Our methodology and experimental setup details are presented in section 4. Section 5 presents the experimental results and provides system level analysis along the micro-architectural characterization of big data applications. Section 6 provides the related work. Lastly, section 7 concludes the paper.

2. Background on Big Data Applications

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 creates opportunities 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

Table 1: Studied Big Data Applications

Workloads	Applications		Data Semantics	Software Stacks
Big Data	Hadoop Micro-benchmarks	WordCount	Generated	Hadoop 1.2.1
		Sort		
		Grep		
		TeraSort		
		TestDFSIO-Write		
		TestDFSIO-Read		
	Graphbuilder [12]		Wikipedia	GraphBuilder 0.0.1, Hadoop 1.2.1
	Collaborative Filtering [13]		MovieLens	Hadoop 1.2.1, Mahout 0.6
	Clustering [13]		UCI Machine learning Repository	
	FP-Growth [13]		Frequent Itemset Mining Dataset Repository [14]	
	SPMF [15]	Eclate		SPMF 0.96
		RuleGrowth		
		GSP		
		SPADE		
Scale-out	Classification [3]		Wikipedia	Hadoop 1.2.1, Mahout 0.6, Memcached server/client 1.4.15, CloudSuite 2.0
	Graph-Anlysis [3]		Twitter data set	
	Data-caching [3]		Twitter data set	
Traditional	Spec2006		Reference input	Spec 2006
	Parsec		Native input	Parsec 2.1

as the four "Vs", shown in Figure 1 [38]: volume, velocity, variety, and veracity. Big data is inherently large in volume. Velocity refers to how fast the data is coming in and to how fast it needs to 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 of data. Traditionally, cloud servers mainly use high performance CPU cores such as Xeon. However, low-power embedded cores such as Atom are gradually entering the server market. Therefore, it is important to characterize emerging big data applications on these two different platforms to understand their computational need and architectural bottlenecks.

3. Dominant Big data Workloads

The studied big data workloads in this paper are representative programs from 15 different domains such as graph mining, data mining, data analysis platform and pattern searching applications, which are frequently used in the real world. We provide these selected applications, along with their particular domain and data type in Table 1.

3.1 Big data Workload

3.1.1 ***Hadoop Microbenchmark.*** 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 has numerous micro-benchmarks from which we have included a combination of I/O intensive and CPU intensive applications as follows:

- *WordCount* reads text files and determines how often the words appear in a set of files. Wordcount is a CPU intensive application [30].
- *Sort* uses the map/reduce framework to sort the input directory in the output directory. The actual sorting occurs in the internal shuffle and sort phase of MapReduce. The data is transferred to reducer that is an identity function. Sort is an I/O intensive application [30].
- *Grep* extracts matching strings provided by user from text files and sorts matching strings by their frequency. Grep is a CPU intensive application.
- *TeraSort* performs a scalable MapReduce-based sort of input data. It first samples the input and computes the input distribution by calculating the quantiles equal to the number of reduces that uses a sorted list of N-1 sampled keys to define the key range for each reduce. TeraGen command generates the large random data for TeraSort. [30].
- *TestDFSIO-write/read* is a storage throughput test that is divided into two parts, TestDFSIO-Write and TestDFSIO-Read to write and read data to/from HDFS, respectively.

3.1.2 ***GraphMining***. Graph construction can be very challenging because of complex iterative and data-dependent nature of the graph. Hadoop is well suited for this task, but requires expertise to handle graph complexities. GraphBuilder addresses this challenge by providing a scalable graph construction software library for Hadoop. GraphBuilder constructs graphs for PageRank and LDA algorithms implemented on PowerGraph [12].

3.1.3 ***Collaborative Filtering (CF) Recommendation*** is a technique used by many recommender systems to predict the preference of users based on their previous rating history [13].

3.1.4 Clustering is one of the fundamental tasks in Data Mining. Clustering assembles data items into groups based on their similar features. We have analyzed meanshift clustering as it is a non-parametric clustering technique that does not require prior knowledge of the number of clusters [13].

3.1.5 ***Association Rule Mining*** is a well-known approach for exploring association between various parameters in large databases. We have analyzed FP (Frequent Pattern)-Growth; a resource intensive program that aims to determine item sets in a group and identifies which items typically appear together [13,14].

3.1.6 ***Sequential Pattern Mining Framework (SPMF)*** is an open-source data mining library written in Java. It offers numerous data mining algorithms for sequential pattern, rule mining and frequent item mining [15] for which we have selected Equivalence Class Transformation (Eclat), RuleGrowth, Generalized Sequential Pattern (GSP) and Sequential Pattern Discovery using Equivalence classes (SPADE).

3.2 Scale-Out Workloads

3.2.1 ***Classification*** technique learns from the existing categorizations and groups the unclassified items to the best corresponding category [3].

3.2.2 ***Graph-analysis*** is performed by implementing the TrunkRank on GraphLab [3]. This application studies the impact of a Twitter user for graph analysis.

3.2.3 ***DataCaching.*** Memcached is a high-performance, general-purpose distributed memory caching system. It uses in-memory key-value storage mechanism for small chunks of arbitrary data API calls [3].

3.3 Traditional CPU Benchmarks

3.3.1 ***SPEC CPU2006*** workloads are industry standard real-life applications designed to stress the CPU, memory subsystem and compiler.

3.3.2 ***PARSEC 2.1*** is an open-source parallel benchmark suite for evaluating multi-core and multiprocessor systems.

4. Measurement tools and Methodology

Figure 2 presents a methodology of our approach. We conduct our study on two state-of-the-art servers, Intel Xeon and Intel Atom. Intel Xeon E5 enclosed with two Intel E5-2420 processors that includes six aggressive processor cores per node with three-level of the cache hierarchy. Intel Atom C2758 has 8 processor cores per node and a two-level cache hierarchy. Table 2 summarizes the key architectural parameters of these two servers. The operating system used is Ubuntu 13.10 with Linux kernel 3.11.

We analyze the architectural behavior using Intel VTune [16], a performance-profiling tool that provides an interface to the processor performance counters. We have used Watts up PRO power meter to measure the power consumption of the servers [17]. Wattsup power meter produces the power consumption profile every one second of an application under

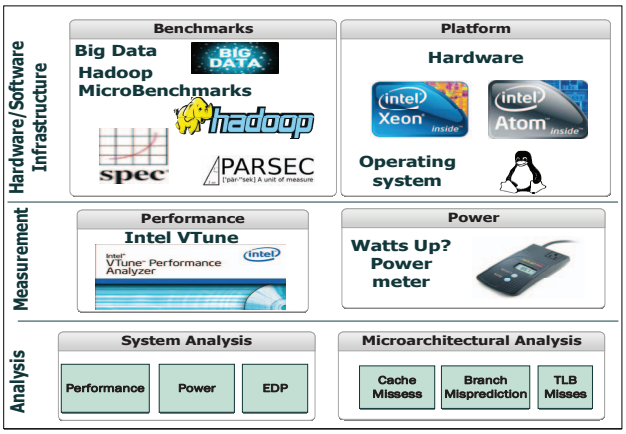


Figure 2: Methodology

test. The power reading is for the entire system, including core, cache, main memory, hard disks and on-chip communication buses. We have collected the average power consumption of the studied applications and subtracted the system idle power to calculate the dynamic power dissipation of the entire energy analysis. The same methodology is used in [1] as well.

We discuss the system-level - performance, power, EDP- and microarchitecture-level analysis - cache misses, branch misprediction and TLB misses- for big data applications and Hadoop micro-benchmarks. In addition, Table 1 shows the datasets used to drive the studies applications.

5. Experimental Results and Analysis

In this section, we discuss the system-level and micro-architecture-level analysis of little and big cores, when running traditional CPU benchmarks, parallel benchmarks, scale-out, and big data applications. Due to space constraints, we are only reporting the average results for SPEC, PARSEC and Scale-Out applications. Moreover, we have conducted the data size sensitivity analysis of Hadoop micro-benchmarks with the dataset of 10MB, 100MB, 1GB and 10GB per node to understand the impact of the size of data per processing node on system-level as well as microarchitecture-level parameters.

5.1 Performance Analysis

In this section, we analyze the performance measurements of big data applications in term of IPC and compare it with the traditional benchmarks. Figure 3.1 presents that the average IPC of big data is 1.65 times lower than the traditional CPU benchmarks on big core and 1.21 times on little core. Therefore, noticeably more performance drop (37%, on average) is observed for big data applications compared to traditional CPU applications when running on big core server compared to little core server. In general, we observe lower IPC in big data applications compared with the traditional benchmarks. Furthermore, little core-based server is experiencing 1.43 times lower IPC in comparison to big core server as Xeon can process up to 4 instructions simultaneously while Atom is limited to 2 instructions per cycle. Figure 3.2 shows the IPC of Hadoop micro-benchmarks for different data sizes. The results are consistent with the results in Figure 3.1 showing lower IPC on little core compared to big core across all data sizes. We also observe that on little core, increasing the data size reduces the IPC since the cache misses increases (mainly Icache miss as will be described in section 5.5.1). Little core, due to its low processing capacity (issue width of 2), cannot hide cache miss penalty as effective as big core. However, on big core while for most cases, increase in data size per node reduces the IPC, there are few exceptions where increasing the data size from 100MB to 1000MB per node increases the IPC. This is mainly due to higher cache locality as a result of larger and more complex cache subsystem in big core, which results in reduction in cache miss rates (more on this in section 5.5.1).

5.2 Power Consumption and Energy-Efficiency Analysis

In this section, we report the power consumption of big data applications and discuss the energy-efficiency analysis to evaluate the trade-off between power and performance.

5.2.1 Power Characterization

Figure 4.1 shows the average dynamic power consumption of the studied applications on big and little core servers. The idle power of the servers is subtracted from the measured (run-time) power. Note that the power results reported are for the entire system, including core, cache, DRAM and on-chip communication buses. Big core consumes on average 35 Watts of dynamic power with the peak of 44 Watts in cluster application. Little core consumes much lower dynamic power as expected, ranging from 0.9 to 6 Watts with an average of 4.8 Watts. Figure 4.2 shows that the power consumption increases as the size of data per node increases in most cases across both big and little architectures. This is more noticeable in little core. While increasing in data size in little core reduces the IPC and therefore core power, it increases cache and off-chip traffic in DRAM and bus subsystem (see LLC MPKI reported in Figure 10). Therefore, for low-end little core where cache, DRAM and off-chip components are dominant power consumer (unlike high performance Xeon core), a clear rise in power consumption is observed as the size of data increases.

5.2.2 Energy-Efficiency Analysis with near Real–Time Processing Constraints

Based on the results of power consumption for both platforms, we have evaluated the trade-off between power and performance by investigating the EDP metric. Furthermore, we have explored the ED^2P and ED^3P to understand the impact of near real-time performance constraints on big data applications and how more constraints on performance affects the choice of most efficient server architecture. Figure 5.1 illustrates EDP, ED^2P and ED^3P ratio for big vs little cores. The ED^XP

Table 2: Architectural Parameters

Processor	Intel Atom C2758	Intel Xeon E5-2420
Operating Frequency	2.40GHz	1.9GHz
Micro-architecture	Silvermont	Sandy Bridge
L1i Cache	32 KB	32 KB
L1d Cache	24 KB	32 KB
L2 Cache	4*1024 KB	256KB
L3 Cache	-	15MB
PageTable	16972 KB	4260 KB
System Memory	8GB	32GB
TDP	20W	95W

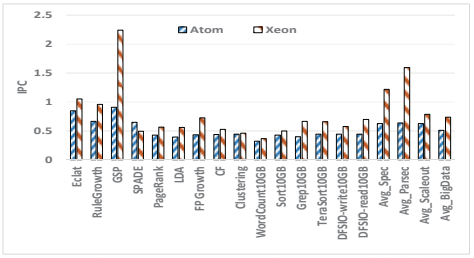
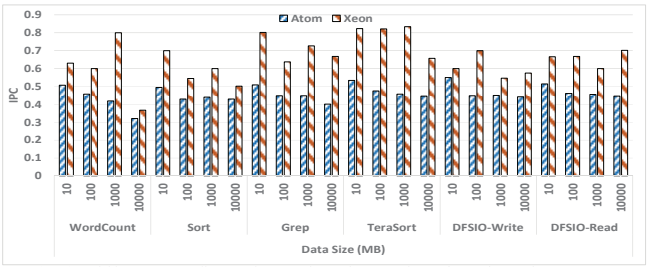


Figure 3: IPC (3.1) Big Data workloads



(3.2) Different configurations of Hadoop micro-benchmarks

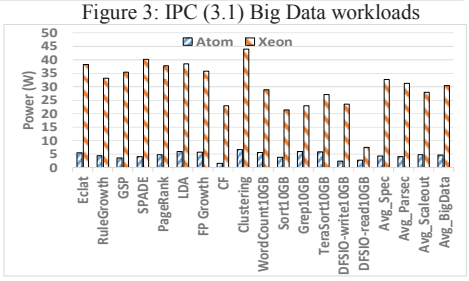
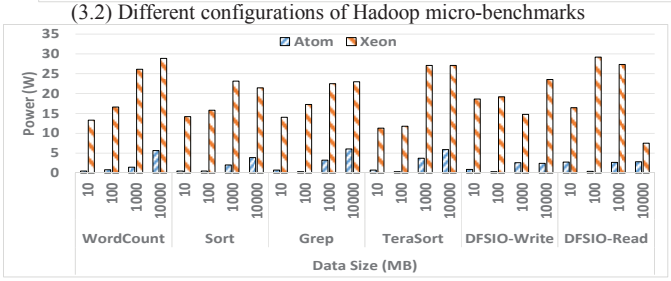
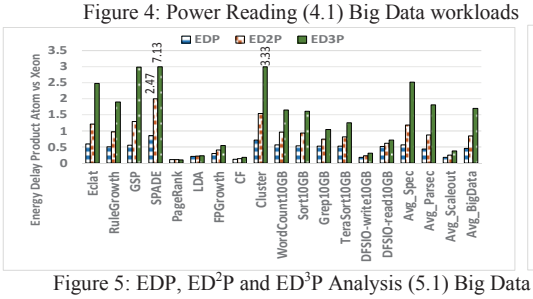


Figure 4: Power Reading (4.1) Big Data workloads



(4.2) Different configurations of Hadoop micro-benchmarks



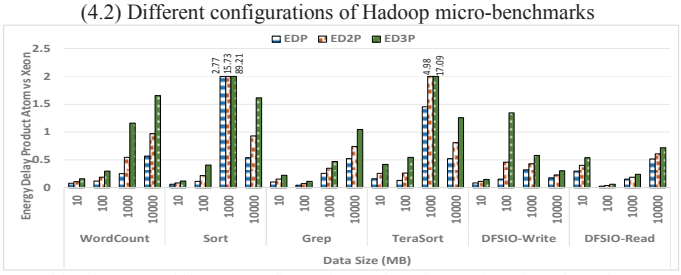


Figure 5: EDP, ED^2P and ED^3P Analysis (5.1) Big Data Workloads (5.2) Different configurations of Hadoop micro-benchmarks

(X=1,2,3) results show that big core-based server is noticeably more efficient for traditional CPU applications compared with big data. Also, interestingly for scale out benchmarks, little core is always more efficient than big core for EDP, ED^2P and ED^3P metrics. For real-world big data applications EDP results show that the little core-based server is almost always a more efficient platform for CPU intensive applications. However, for heavy I/O intensive applications such as sort and terasort, for large data sizes (10000MB), the big core becomes more efficient than the little core in terms of EDP. Complex memory subsystem in big core along with higher processing capacity (2X more than little core) allows big core to be more effective in hiding the cost of high I/O communication in these applications and can explain why big core-based server is more efficient. However, with more near real-time performance constraints, i.e. for ED^2P and ED^3P metrics, big core becomes more efficient compared to little cores across most applications. Figure 5.2 presents the data sensitivity analysis of Hadoop micro-benchmarks. The increase in the data size, progressively makes the big core more efficient compared with little core, however the point where big core becomes more efficient than little core varies across applications and depends on the data size and the performance constraint.

Observation. The results illustrate that little core server is more efficient in terms of EDP for big data applications with the smaller data sizes. However, as the size of data increases and with more performance constraints big core server becomes more efficient.

5.3 DPS-DPJ Analysis

In this section, we evaluate the data processing capability of big and little core-based server for various data sizes in Hadoop micro-benchmarks. We report the data processed per second (DPS) and the data processed per joule (DPJ) metrics to compare the data processing capability and efficiency of the two server architectures.

The results are reported in Figure 6.1 and Figure 6.2. For most applications, on both big and little platforms with an increase in the data size the DPS first rises rapidly to a peak and then declines slightly. The data size at which the peak DPS occurs varies across applications and architectures. The two metrics, DPS and DPJ, also help to guide MapReduce scheduling decision. The peak DPS occurs in terasort and sort at only 1000MB of size, while in other applications occurs in at least an order of magnitude larger data size. The reason is that sort and terasort are I/O intensive applications and the rise in data size exacerbates the I/O cost to an extent that it diminishes the benefit of high processing capacity. The DPS difference between big and little cores-based server is becoming larger for CPU intensive applications such as grep and wordcount as the size of data increases. However, this is not the case for I/O intensive applications such as sort and DFSIO-read as the I/O cost becomes the dominant performance bottleneck and the processing capacity of the processor; i.e. big vs. little become a less important factor. Overall, for small data size, below 1000MB per node, the two architectures, big and little, have almost similar processing capacity in terms of DPS, and it is only for large data sizes that the DPS gap between the two becomes clear.

Similar to DPS, for DPJ, in all applications we observe a rise in data processing efficiency on big core as the size of data increases. However, in I/O intensive applications such as terasort, DFSIO-read and write, the rise in DPJ on Xeon is

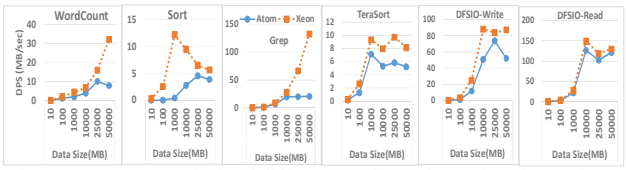


Figure 6 .1: DPS Analysis of different configurations of Hadoop micro-benchmarks

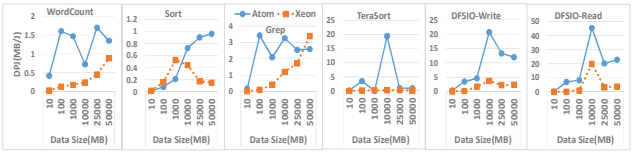


Figure 6.2: DPJ Analysis of different configurations of Hadoop micro-benchmarks

insignificant. For CPU intensive applications including wordcount and grep there is a significant rise in DPJ on Xeon as the size of data increases.

Overall, for I/O intensive applications such as sort, terasort, DFSIO-write and DFSIO-read, Atom-based server is noticeably more efficient than big Xeon. However, in CPU intensive micro-benchmarks, *WordCount and Grep*, the DPJ gap between big and little core-based server reduces with the increase in data size. It is also interesting to observe that the DPJ of Xeon can exceed Atom in a number of applications and across a number of different data sizes.

Observation-The results illustrate that the choice of big vs little core-based servers in terms of DPS and DPJ analysis are closely decided by the application type and the size of data.

5.4 Performance Hotspot and Post-Acceleration CPU code Characterization

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. In addition to big, medium, and small cores, the integration of domain-specific accelerators, such as GPUs and FPGAs has become extensive.

To find out the right server architecture for big data processing, it is important to understand how deploying an accelerator, such as FPGA, would necessitate adapting the choice of CPU. The post acceleration code characteristics are important to find the right architecture for efficient processing of big data applications. In this section, we analyze the choice of big vs. little core-based server for the code that remains for the CPU after acceleration, compared with the choice of big vs. little before acceleration.

A key research challenge for heterogeneous architecture that integrates CPU and accelerator such as FPGA is workload partitioning and mapping of a given application (which is alternatively referred to as scheduling) to CPU and FPGA for power, performance, and QoS. This is commonly referred as hardware and software partitioning. A common method for HW/SW partitioning is to profile the application to find the performance hotspot region. These regions are candidates for FPGA acceleration, as long as the overhead of communication with CPU is not significant [18].

To perform hotspot analysis on big data applications, we use Intel Vtune to select the common hotspot modules of the applications running on big and little cores. First, we identify and analyze hotspot modules based on their execution time. Figure 7.1 shows the common hotspot modules of big data applications and Figure 7.2 presents Hadoop micro-benchmark hotspots with a data size of 1GB on Big and Little cores, respectively. Map and Reduce tasks represent the computation part to perform the task, such as grep, sort and etc. Libz is performing the compression and decompression task (library) for the Hadoop workload. Module dynamic contains hotspot functions such as java-finalize to perform the completion tasks of an object. Compiled java code includes the java.lang.string class to represent character strings along with the system.array, copy, math, abstractSringBuilder and object classes. Libpthread contained functions like mutex lock/unlock for the thread creation and protection and cond_wait functions to block on a condition variable.

We have also collected the IPC for each of these hotspot functions. Due to space limitation, we do not show the details of the IPC results. The results show that the performance gap between big and little cores for Map and Reduce task is 2X. This large gap shows that big core has a clear advantage over little core to run these hotspot functions. Since the hotspot functions and their corresponding libraries are taking up most of execution time, they are candidate for acceleration, for instance with offloading mapper and reducer tasks to FPGAs [19, 20, 21, 25, 28]. Several recent works have demonstrated the potential of offloading map and reduce tasks to FPGA platforms [22, 23, 24, 41].

To analyze post-accelerated code and the choice of server architecture, we assume map and reduce tasks are offloaded to an accelerator [22, 23, 24]. We do not make any assumption about the speedup gain on accelerator nor the cost of offloading to the accelerator. By taking out the time it takes to run hotspot function, we can study the remaining modules that are left for the big or little core-based server to run.

Figure 8 shows the impact of post-accelerated code by investigating the speed up - migrating from Atom to Xeon before and after acceleration. We report the little vs big core speed up in terms of

$$Speed\ up = \frac{(Exectime_{Atom}/Exectime_{Xeon})_{remaining\ code\ after\ acceleration}}{(Exectime_{Atom}/Exectime_{Xeon})_{entire\ application}} \quad Equation\ 1$$

$(Exectime_{Atom} / Exectime_{Xeon})_{remaining\ code\ after\ acceleration}$ represents the speed up obtained by migrating the post-accelerated code from Atom to Xeon.

$(Exectime_{Atom} / Exectime_{Xeon})_{entire\ applications}$ represents the speed up obtained by migrating the application from Atom to Xeon before acceleration.

Using equation 1, we can evaluate the impact of Atom over Xeon speedup gain after acceleration compared to speed up before acceleration. Most of the micro-benchmarks in Figure 8 have speed up less than 1 which indicates that speedup of Atom over Xeon after acceleration reduces compared to speed up before acceleration. We have also observed that most micro-benchmarks, with the increase in data size the speedup after acceleration reduces compared to the speedup before acceleration. However, there are several exceptions to this trend, including GSP, RuleGrowth, Ecalt, *WordCount, and Spade* where post acceleration code achieves

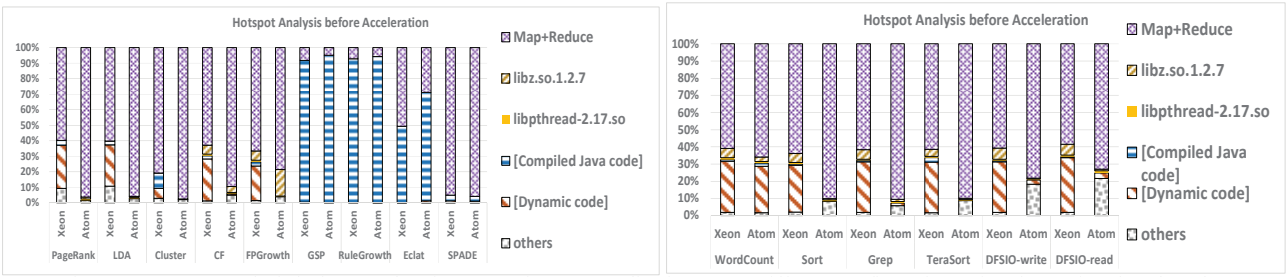


Figure 7. Hotspot Analysis before acceleration (7.1) Big Data Applications (7.2) Different configurations of Hadoop micro-benchmarks

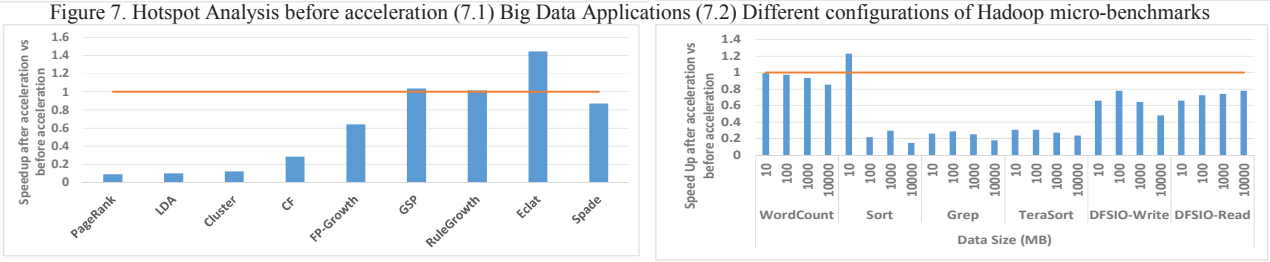


Figure 8: Speed up of Atom vs Xeon before and after acceleration (8.1) Big Data applications (8.2) Different configurations of Hadoop micro-benchmarks

higher speed up on Xeon over Atom compared with pre-acceleration code.

Overall, Xeon provides a lower execution time, however, if speedup after acceleration is very small then considering the power consumption of Xeon, Atom-based will be a more efficient server to execute the post-accelerated code.

Observation: We study how offloading hotspot map and reduce tasks to an accelerator such as FPGA affects the choice of big vs. little core-based server for processing. The results show that the choice of big vs. little before and after accelerations is different. While most benchmarks clearly favor little core post acceleration, in several applications post accelerated code show higher speed up on big core-base server over little core-base server compared to pre-acceleration.

5.5 Microarchitecture-Level Analysis

To provide insight into whether current server design based on big and little core architectures requires improvement in their microarchitecture parameters for efficient big data processing, we perform a comprehensive microarchitecture characterization and compare the results with traditional CPU, PARSEC, and scaleout applications. Microarchitecture-level characterization of big data applications on big and little core architectures play a crucial role in identifying directions where microarchitecture improvement is required. The details of the observed parameters are explained below:

5.5.1 Instruction cache misses

Frequent cache misses decreases the fetch rate and introduces stalls in the frontend of the pipeline. Fetch rate reduction constraints the number of instructions that frontend delivers to the backend for ILP extraction and therefore increases the chance of the whole pipeline to be stalled.

Figure 9.1 presents the L1 Instruction cache misses per kilo instructions (MPKI) on big and little cores. The average I-cache MPKI of big data applications is at least 2.6 times higher than the traditional benchmarks and 1.86 times higher than the scale-out applications on Atom, while on Xeon it is 3.8 times higher than traditional benchmarks and 4.76 times higher than scale-out applications. In comparison to Xeon, Atom shows two times higher I-cache MPKI in big data applications. Figure 9.2 shows data sensitivity analysis of Hadoop micro-benchmarks. Data sensitivity analysis results show an increase in I-cache MPKI on both Xeon and Atom show an increase in I-cache MPKI on both Xeon and Atom as the size of data increases.

Observation. The general observation is that the instruction working sets of most big data workloads exceed the L1 capacity of both Xeon and Atom processor. Fast cache access demand restrains the architectural designer to increase the instruction cache size. However, Xeon and Atom comparison shows that a three-level compared to a two-level instruction cache hierarchy just slightly mitigate instruction cache MPKI. Increase in data size results in increasing the number of dynamic instructions (as the number of iterations in many loops in the code increases) leading to higher L1 instruction cache misses. Current prefetchers rely on the repetitiveness and sequential behavior of the instructions to predict the same sequence in the future. Other possible factors leading to a high I-cache misses are the huge code size and deep software stacks, and relatively large number of system calls in big data workloads. This behavior implies that higher I-cache performance is demanded for big data workloads. Moreover, low-end microarchitecture such as little Atom for big data processing needs to include the instruction prefetchers that can predict complex patterns with the advance replacement policy to eliminate the wasted cycles caused by front-end stalls.

5.5.2 LLC cache misses

Intel Xeon has a three-level cache hierarchy and Intel Atom has a two-level hierarchy to reduce the gap between processor and memory speed. In this study, we have analyzed the LLC of both processors; L3 cache of Xeon and L2 cache of Atom. Figure 10.1 shows the LLC MPKI on Atom and Xeon for big data applications along the average LLC MPKI of SPEC, PARSEC and Scale-Out applications. The average LLC data cache MPKI of Big data is at least 2.5 times higher than the traditional benchmarks and 3.8 times higher than Scale-Out applications on Xeon. In contrast, Atom shows an opposite behavior; SPEC experiences 3.25 times higher data MPKI than big data on Atom. Moreover the average LLC data cache MPKI of big data on Atom is 1.8 times higher than Xeon. Figure 10.2 shows the data sensitivity analysis of Hadoop

micro-benchmarks on Xeon and Atom. Data MPKI in Atom increases with the increase in data size, however Xeon shows no clear trend.

Observation. We conclude that big data applications have relatively good data locality and there are fewer computation operations to memory access compared with traditional benchmarks corroborating the observation in [3]. Furthermore, this result shows that while three-level cache hierarchy with the large size in Xeon processor has an advantage over Atom, a small 4x1MB LLC cache in Atom is sufficient for many big data applications to provide relatively low MPKI rate. In addition, we observed that while in most cases increasing data size increases LLC MPKI, there are exceptions that require careful consideration when choosing the size of data to be allocated to each node to reduce LLC MPKI.

5.5.3 Branch Misprediction

Figure 11.1 and 11.2 presents the branch misprediction results. The average misprediction in big data is 9.3%, while that of SPEC, PARSEC and Scale-Out is 5.8%, 8.01% and 8.66% On Atom processor, respectively. In addition, on Xeon,

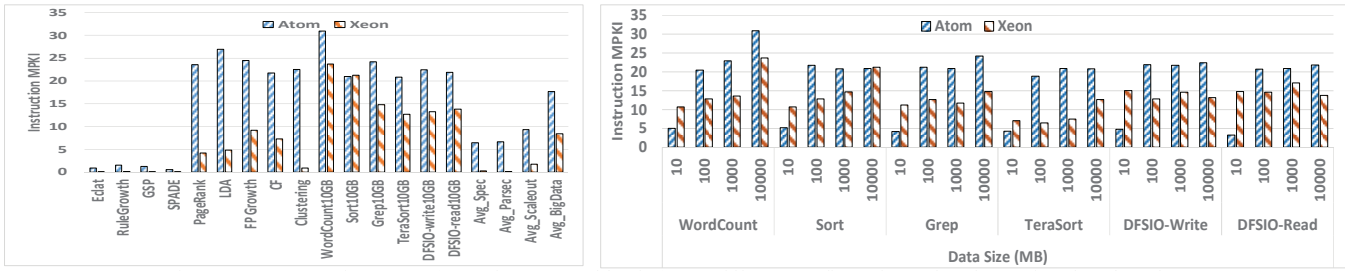


Figure 9: L1 Instruction MPKI (9.1) Big Data workloads (9.2): Different configurations of Hadoop micro-benchmarks

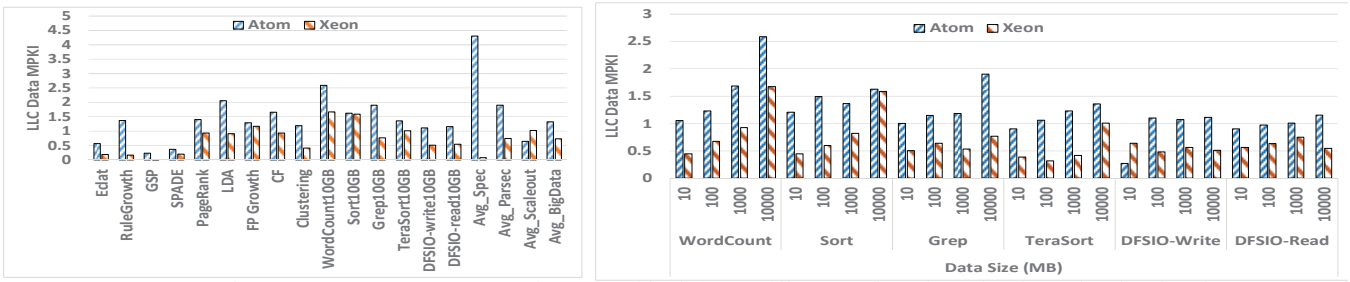


Figure 10: LLC Data MPKI (10.1) Big Data workloads (10.2) Different configurations of Hadoop micro-benchmarks

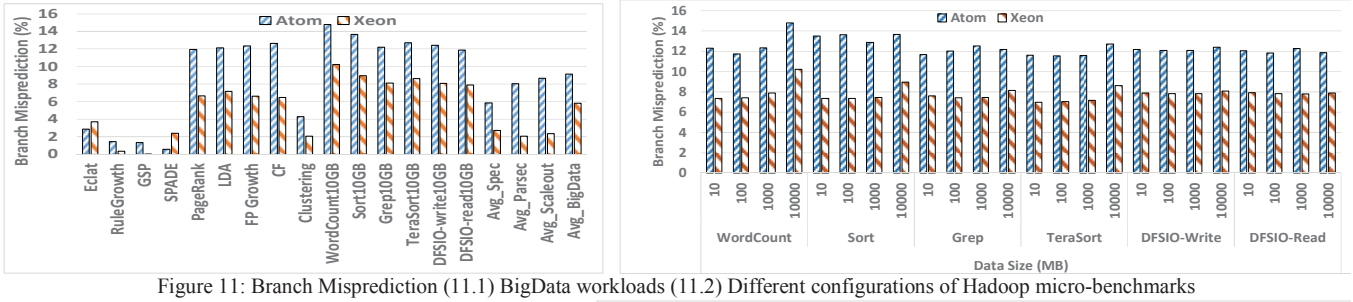


Figure 11: Branch Misprediction (11.1) BigData workloads (11.2) Different configurations of Hadoop micro-benchmarks

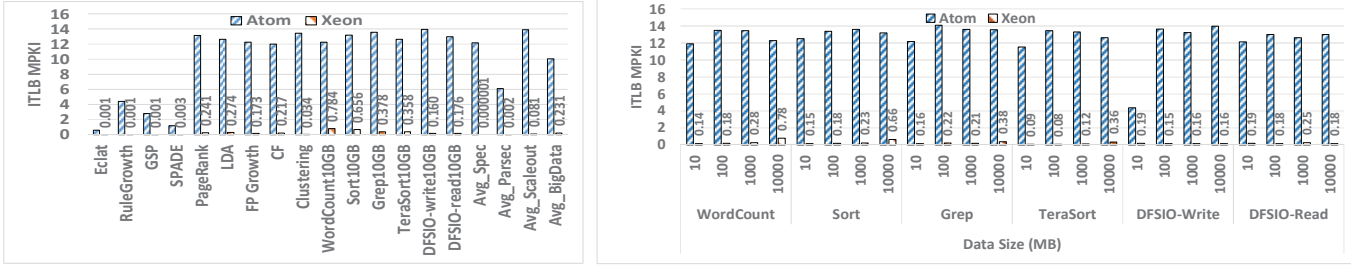


Figure 12: ITLB MPKI (12.1) Big Data workloads (12.2) Different configurations of Hadoop micro-benchmarks

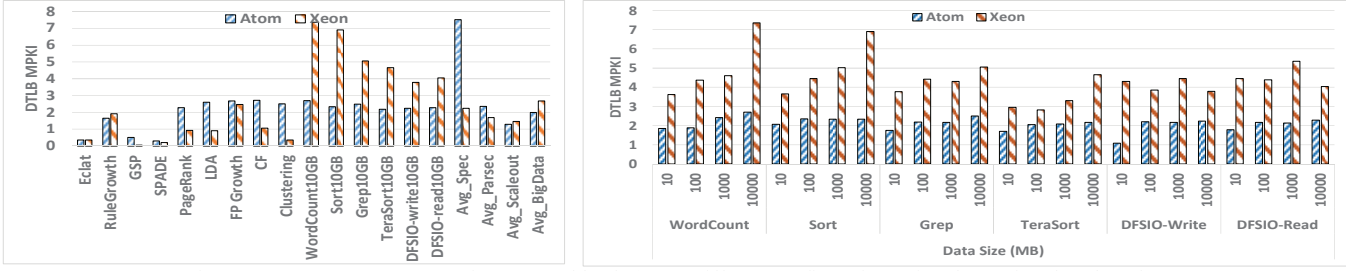


Figure 13: DTLB MPKI (13.1) Big Data workloads (13.2) Different configurations of Hadoop micro-benchmarks

average misprediction of big data is 5.9%, while that of SPEC, PARSEC and Scale-Out is 2.7%, 2% and 2.3% respectively. This shows that branch misprediction rate in big data is at least 1.2 times higher on Atom and 2.13 times higher on Xeon compared with traditional benchmarks. In comparison, Atom experiences 1.5 times higher branch misprediction than Xeon processor. Figure 11.2 results show that the branch misprediction rate does not change noticeably with changing the data size.

Observation. Traditional CPU and parallel benchmarks mainly having instruction working sets with tight loops (as in matrix algebra), making them easy for the branch predictor to predict correctly and reduce the misprediction rate. In big data workloads, however, loops are seldom, and instead they fetch record (LD), Match Key (CMP), and non-loop branches (Branch to handler or BC) operations. This results in higher branch misprediction rate and affects the application performance by creating stalls in the pipeline.

5.5.4 TLB misses

TLB (Translation look-aside buffer) misses are costly, in terms of both performance as well as power, taking up hundreds of cycles to respond. We have reported the Instruction TLB (ITLB) and Data TLB (DTLB) MPKI in Figure 12 and Figure 13. Our results reveal that on Xeon big data has the highest ITLB MPKI with the average of 0.23 while that of SPEC, PARSEC and Scale-Out is orders of magnitude lower with an average of 7.44E-07, 0.0019, and 0.081 respectively. The average DTLB MPKI of big data is just slightly higher than traditional benchmarks on Xeon, however on Atom SPEC has noticeably higher DTLB MPKI reading than others. On Xeon big data has the highest ITLB and DTLB MPKI compared with the traditional benchmarks. In contrast to this behavior SPEC is experiencing the highest ITLB and DTLB MPKI on Atom. Moreover, Atom incurs large ITLB MPKI as compared to Xeon, but lower DTLB MPKI in comparison to Xeon. Figure 12.2 and Figure 13.2 are showing the data sensitivity analysis of micro-benchmarks with respect to ITLB MPKI and DTLB MPKI. Increasing data size clearly increases the DTLB MPKI across both Atom and Xeon in most benchmarks, however, it does not have a noticeable impact on ITLB MPKI.

Observation. Overall, on Xeon, big data applications have an order of magnitude lower ITLB miss rate compared to Atom, however on Xeon they have an order of magnitude higher ITLB miss rate compared to the traditional CPU benchmark. While the results show TLB miss overhead management is important in both Atom and Xeon for big data applications, the large gap between big data applications and traditional CPU benchmark on Xeon is calling for a big-data specific TLB management technique. Also the results show that the size of data affects DTLB miss rates in both architectures.

6. RELATED WORK

Recently, there have been a number of efforts to benchmark and characterize big data and cloud-scale applications, mainly on state-of-the-art high performance server platform. In general, there are two major approaches for benchmarking big data: A system benchmarking and a component benchmarking. A system benchmark is an end-to-end benchmarking which includes the entire database and application software stack, including data preparation, data aggregation and data analytics. A component benchmark encloses only a portion of the entire end-to-end system [29].

The most prominent big data benchmarks, include HiBench, Scale-Out, BigDataBench, CloudCmp, and LinkBench. HiBench [30] is a benchmark suite for Hadoop MapReduce. CloudCmp [31] use a systematic approach to benchmark various components of the cloud to compare cloud providers. LinkBench is a real-world database benchmark for social network applications [32]. The Transaction Procession Performance Council (TPC) has released a number of benchmark suites in recent years, including TPC-C, TPC-E, and TPC-DS for online transaction processing. BigDataBench [2] was released very recently and includes online service and offline analytics for web service applications. BigBench [4] is a new big data benchmark that adopts TPC-DS as its basis and expands it for offline analytics on Xeon high performance server. The CloudSuite [3, 4] benchmark was developed for Scale-Out cloud workloads and mainly includes small data sets, e.g., 4.5 GB for Naïve Bayes.

Several prior researches have characterized traditional CPU and parallel applications such as SPEC2006, PARSEC, and NAS on high performance server-class processors [39]. It is important to also compare the characteristics of big data application with these traditional benchmark suites. We have included the SPEC CINT2006, SPEC CFP2006 and PARSEC 2.1 benchmarks for the comparison with BigData Workloads.

This work is different from all above benchmarking and characterization work as it perform a comprehensive system-level (power, performance, ED^XP, DPS and DPJ) and microarchitecture-level(cache miss, TLB miss, branch misprediction) analysis of various big data applications and big data micro-benchmarks on two substantially different platforms one with high performance big core and another with low power little core to understand which of these two architectures is the choice for efficient big data processing.

There have been also a number of research into application-specific [34, 40] and domain-specific accelerators [35, 36, 37]. Using tightly integrated FPGA [33] with CPU, and GPU with CPU [27], to accelerate big data processing have been proposed in recent work. While deploying programmable accelerator is a new and hot research topic, there has been little attention paid to how CPU designs should be adapted to this change. To the best of our knowledge, the only work on this topic is by Arora [29], which studied the role of the CPU for a CPU+GPU architecture. They concluded that, in a CPU+GPU architecture, the CPU is running a code that is significantly different from a CPU-only code. They found that the post-GPU code has a lower ILP, higher branch miss prediction rate, and larger number of load and stores, and benefits less from multiple cores, as there is less TLP after GPU offloading. In this paper, we demonstrated how deploying accelerator such as FPGA for big data affects the choice of big vs. little core for efficient processing.

7. CONCLUSIONS

In this paper, we present a comprehensive system and micro architecture-level analysis of big data applications on

two distinct server platforms; the conventional approach, a high performance big Xeon core; and the new trajectory in server design, a low power little Atom core, which advocates the use of a low-power core to address the power challenge.

The characterization results show significantly larger performance drop (37%, on average) for big data applications compared to traditional CPU applications when running on big core server compared to little core server. Big core-based server provides a high performance, compared to little core, however, it is not as power efficient. Little core-based server is more efficient in terms of EDP for big data processing with small data sizes. However, as the size of data increases and with performance constraints, big core becomes an efficient choice. The analysis of data processing capability and efficiency of big data applications illustrates that the choice of big core vs. little core-based server in terms of data processing per second and data processing per joule is closely decided by the application type, size of data, and computational and I/O intensity of the application.

In addition, we performed the post-acceleration CPU code analysis to find out the most efficient server architecture to process the remaining code of big data applications. The results show that there is a difference between the choice of big vs. little core-based server before and after accelerations. While most benchmarks clearly favor little core post acceleration, several applications show higher speed up on big core over little core post acceleration compared to pre-acceleration.

To provide insight on whether current server design based on big and little core architectures requires improvement in their microarchitecture parameters for efficient big data processing, we perform a comprehensive microarchitecture characterization and compare the results with traditional Spec, PARSEC, and scaleout applications. Our analysis indicates that the size of data has a non-trivial impact on several micro-architecture parameters. Moreover, results show that while a small 4x1MB two-level data cache is sufficient for big data applications on little core the instruction cache hierarchy pipeline needs improvement. Also little core needs architectural improvement in instruction TLB miss overhead management as well as branch predictor. Furthermore, the analysis shows that the deep software stack of big data applications, along with the excessive non-loop branches, affects L1 cache hit rate and branch predictor accuracy in both big and little cores. Moreover, big data applications require efficient instruction prefetchers to predict complex patterns and sophistication branch predictor to handle the unknown control flow.

References

[1] Blem, E., et al., "Power struggles: Revisiting the RISC vs. CISC debate on contemporary ARM and x86 architectures," In HPCA2013, IEEE 19th International Symposium on (pp. 1-12).

[2] Gao,W. et al.,"BigDataBench: a Big Data Benchmark Suite from Web Search Engines,".ASBD 2013 in conjunction with ISCA 2013

[3] Ferdman, M., et al. "Clearing the clouds: a study of emerging scale-out workloads on modern hardware," ACM SIGARCH Computer Architecture News40.1 (2012): 37-48.

[4] Ghazal, A. et al., "Bigbench: Towards an industry standard benchmark for big data analytics," In: ACM SIGMOD Conference (2013)

[5] Kontorinis, V., et al., "Managing distributed UPS energy for effective power capping in data centers," In 39th ISCA 2012.

[6] Gutierrez, A. et al. "Integrated 3D-stacked server designs for increasing physical density of key-value stores." Proc. of 19th ASPLOS.ACM, 2014

[7] Reddi, V. J., et al., "Web search using mobile cores: quantifying and mitigating the price of efficiency," ACM SIGARCH Computer Architecture News38.3 (2010): 314-325.

[8] Homayoun, H., et al., "Dynamically heterogeneous cores through 3D resource pooling," In HPCA 2012.

[9] Andersen, D. G. et al. "FAWN: A Fast Array of Wimpy Nodes,"In the Proceedings of ACM SIGOPS 22nd SOSP, pages 1–14, 2009.

[10] Hardavellas, Nikos, et al. "Toward dark silicon in servers." IEEE Micro 31.EPFL-ARTICLE-168285 (2011): 6-15.

[11] Kontorinis, V., et al., "Enabling Dynamic Heterogeneity Through Core-on-Core Stacking," The 51st Annual DAC 2014

[12] Willke, T.L., et al. "GraphBuilder–A Scalable Graph Construction Library for Apache™ Hadoop™," Big Learning WS at NIPS, 2012.

[13] Apache Mahout: scalable machine-learning and data-mining library

[14] Frequent Itemset Mining Dataset Repository; http://fimi.ua.ac.be/data/

[15] SPMF; http://www.philippe-fournier-viger.com/spmf/index.php?

[16] Intel VTune Amplifier XE Performance Profiler. http://software.intel.com/en-us/articles/intel-vtune-amplifier-xe/

[17] WattsUpPro power meter https://www.wattsupmeters.com/secure/index

[18] Nilakantan, S., et al. "Platform-independent analysis of function-level communication in workloads." IISWC, IEEE, 2013.

[19] Accelerating Hadoop* Applications Using Intel® QuickAssist Tech., [online]

[20] Neshatpour, K, Malik, M., Ghodrat, M. A., Homayoun, H., "Accelerating Big Data Analytics Using FPGAs," IEEE FCCM 2015

[21] James T Kukunas, et al. "High Performance ZLIB Compression on Intel®Architecture Processors," White paper, April 2014.

[22] Shan, Y., et al. "FPMR: Mapreduce framework on FPGA," in Proc Annual ACM/SIGDA Int Symp Field Programmable Gate Arrays, 2010

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

[24] Z. Lin and P. Chow, "Zcluster: A zynq-based hadoop cluster," in Int. Conf. FPT, Dec 2013, pp. 450–453.

[25] Neshatpour, K, Malik, M., Homayoun, "Accelerating Machine Learning Kernel in Hadoop Using FPGAs," In 15th IEEE/ACM CCGrid 2015.

[26] Absalyamov, I., et al., "High-Performance XML Twig Filtering using GPUs," ADMS@ VLDB. 2013.

[27] Baru, C., et al. "Setting the Direction for Big Data Benchmark Standards",Lecture Notes in Computer Science

[28] Khavari Tavana, et al., "Energy-efficient mapping of biomedical applications on domain-specific accelerator under process variation," ISLPED'14

[29] Arora, Manish, et al. "Redefining the Role of the CPU in the Era of CPU-GPU Integration," Micro, IEEE 32.6 (2012): 4-16.

[30] Huang, S., et al. "The HiBench benchmark suite: Characterization of the MapReduce-based data analysis," In the proc. of 26th ICDEW, 2010

[31] Li,A.,et al. "CloudCmp: comparing public cloud providers," ACM,'10

[32] Armstrong, et al. "Linkbench: a database benchmark based on the facebook social graph," Proceedings of the ACM SIGMOD, 2013.

[33] Xi Luo, Walid A. Najjar, Vagelis "Hristidis: Efficient near-duplicate document detection using FPGAs," BigData 2013

[34] YU, P. et al. "Disjoint pattern enumeration for custom instructions identification," In Proceedings of the FPL'07, 273–278.

[35] ARNOLD, M. et al. "Designing domain-specific processors," In Proceedings of the 9th CODES. ACM 2001.

[36] Li, T., et al. "Fast enumeration of maximal valid subgraphs for custom-instruction identification," Proceedings of the CASES. ACM, 2009.

[37] Arora, N, et al. "Instruction selection in asip synthesis using functional matching," VLSI Design, 2010.

[38] http://www.chipestimate.com/tech-talks/2013/07/16/Cadence-5-Emerging-DRAM-Interfaces-You-Should-Know-for-Your-Next-Design-

[39] T. K. Prakash et al. "Performance Characterization of SPEC CPU2006 Benchmarks on Intel Core 2 Duo Processor," In ISAST 2008.

[40] YU, P., et al. "Scalable custom instructions identification for instruction-set extensible processors," In Proc. of the CASES'04. ACM, New York.

[41] Neshatpour, K, Malik M., Ghodrat, M. A., Sasan, A., Homayoun, H., "Energy-Efficient Acceleration of Big Data Analytics Applications Using FPGAs," BigData 2015