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

MARIA MALIK, SETAREH RAFATIRAD, and HOUMAN HOMAYOUN,
George Mason University

The rapid growth in data yields challenges to process data efficiently using current high-performance 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. Low-power embedded cores in servers such as little Atom have emerged as a promising solution to enhance energy-efficiency to address these challenges. Therefore, the question of whether to process the big data applications on big Xeon- or Little Atom-based servers becomes important. In this work, through methodical investigation of power and performance measurements, and comprehensive application-level, system-level, and micro-architectural level analysis, we characterize dominant big data applications on big Xeon- and little Atom-based server architectures. The characterization results across a wide range of real-world big data applications, and various software stacks demonstrate how the choice of big- versus little-core-based server for energy-efficiency is significantly influenced by the size of data, performance constraints, and presence of accelerator. In addition, we analyze processor resource utilization of this important class of applications, such as memory footprints, CPU utilization, and disk bandwidth, to understand their run-time behavior. Furthermore, we perform micro-architecture-level analysis to highlight where improvement is needed in big- and little-core microarchitectures to address their performance bottlenecks.

CCS Concepts: • **Computer systems organization → Architectures**; **Multicore architectures**; • **Hardware → Power estimation and optimization**;

Additional Key Words and Phrases: Performance, power, characterization, big data, high-performance server, low-power server, accelerator

ACM Reference format:
Maria Malik, Setareh Rafatirad, and Houman Homayoun. 2018. System and Architecture Level Characterization of Big Data Applications on Big and Little Core Server Architectures. *ACM Trans. Model. Perform. Eval. Comput. Syst.* 3, 3, Article 14 (July 2018), 32 pages.
https://doi.org/10.1145/3229049

1 INTRODUCTION

Big data technology is driving an era of unprecedented innovation for information retrieval. The rapid growth of data creates challenges to process them for analytics using existing computing solutions. The world of big data is constantly changing and producing huge amounts of data that

This work is supported by the Nation Science Foundation under grant no. CNS 1526913.
Authors' addresses: M. Malik and H. Homayoun, Engineering Building 3223, Department of Electrical and Computer Engineering, 4400 University Dr, Fairfax, VA 22030; emails: {mmalik9, hhomayou}@gmu.edu; S. Rafatirad, Engineering Building: 5304, Department of Information Sciences and Technology, 4400 University Dr, Fairfax, George Mason University; email: srafatir@gmu.edu.

https://doi.org/10.1145/3229049

provides opportunities to find insights in current data and gives answers to the questions that were previously conceded beyond our ability to process. 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 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. First, this trend increases the power consumption of servers, which is the main obstacle for their scalability (Kaushik et al. 2010). Second, the infrastructure cost, which contributes to major financial burden (Barroso et al. 2013), results in prolonging the break-even point of data center profit. 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 (Ferdman et al. 2012; Ghazal et al. 2013; Kontorinis et al. 2012; Gutierrez et al. 2014). Current server designs, based on commodity high-performance and power-hungry processors are not an efficient solution to deliver green computing in terms of performance/watt (Malik et al. 2016; Homayoun et al. 2012; Andersen et al. 2009). This new set of challenges is necessitating a change in the direction of server-class microarchitecture to improve their computational efficiency. Therefore, the embedded processors that have been designed and developed based on energy-efficiency constraints are finding their way in server architectures. Microservers employ embedded low-power processors as the main processing unit. These platforms are shown to be a promising solution to enhance the energy efficiency and to reduce the cost. We believe this is the right time to answer the important question of whether low-power embedded like cores are a good fit to process big data applications energy-efficiently (Malik et al. 2015a). In this article, we will answer this question by identifying the right computing platform for Big Data Analytics processing that can provide a balance between processing capacity and power efficiency. Emerging big data applications, in particular, from web service domains, share many inherent characteristics that are fundamentally different from traditional desktop, parallel, and scale-out applications (Gao et al 2013). Big data analytics applications in these domains heavily rely on big-data-specific deep machine-learning and data-mining algorithms. Additionally, the significant interaction of running complex database software stack with I/O and OS, exhibits high computational intensity, memory intensity, I/O intensity and control intensity behavior. Therefore, 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. In this work, we show that while high-performance big-core servers 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 article, we also present a comprehensive system, application and micro-architectural level analysis of big data applications on two very distinct server-class micro-architectures; Intel Xeon and Intel Atom. These two types of servers represent two schools of thought on server architecture design: Xeon, a big core for high-performance servers, and Atom, a little core that advocates the use of a low-power server to address the dark silicon challenge (Reddi et al. 2010; Homayoun et al. 2012; Andersen et al. 2009; Hardavellas et al. 2011; Kontorinis et al. 2014).

To compare the choice of big-core vs. little-core servers, we use power consumption and performance (in terms of execution time) metrics. In addition to power and performance, we have also performed the Energy-Delay^X (ED^{X}P) analysis to evaluate the trade-off between power and performance (Malik et al. 2015b; Sayadi et al. 2018). The analysis illustrates how near real-time performance constraint 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 amounts of data is a prime target, we have therefore 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 examines how the choice of big vs. little core servers introduces trade-off in performance, power, energy-delay, and processing capacity for efficient and near real-time processing of big data applications.

Additionally, 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 (Hardavellas et al. 2011). 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-core servers. 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 when the hotspots are offloaded to an accelerator, compared with the choice of big vs. little core 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. Furthermore, we perform comprehensive micro-architecture level analysis to provide insight into whether current design of big and little core servers requires improvement in the microarchitecture parameters for efficient big data processing.

Contributions: This article 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: Big cores Intel Xeon and Little cores Intel Atom. We compare the results with traditional CPU, parallel, and scale-out applications.
- We evaluate resource utilization profiles, including total CPU utilization, iowait CPU utilization, memory footprint, and disk bandwidth, to understand the runtime behavior of big data applications and to classify them into computational-intensive, I/O-intensive, or hybrid on big- and little-core-based servers. This is done at coarse-grained (entire application run-time) and fine-grained levels (one second interval). We compare the results with the traditional, parallel, and scale-out applications.
- We demonstrate how the size of data, type of application, and performance constraints affect the choice of big- vs. little-core-based servers for energy-efficient big data processing.
- We analyze the performance of hotspot tasks involved in an end-to-end big data analytics platform 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 show how offloading hotspot tasks such as map and reduce to an accelerator 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-core servers when running big data applications.

The rest of the article is organized as follows. Section 2 provides background on big data. Section 3 describes the studied big data, scale-out, 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 with micro-architectural characterization. Section 6 provides the related work. Section 7 concludes the article.

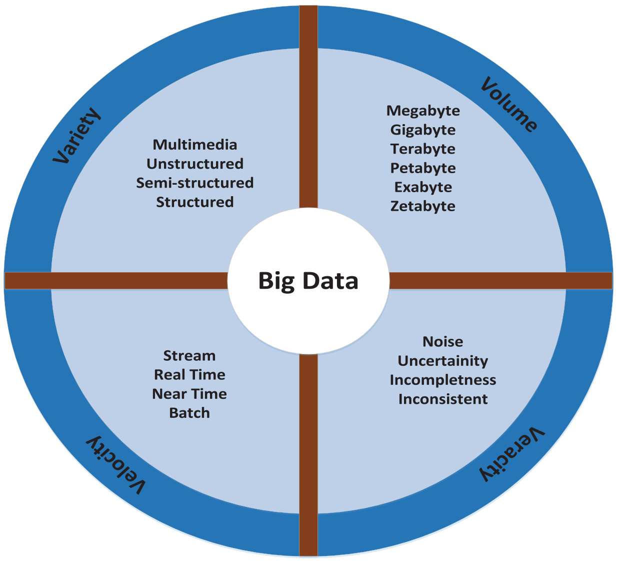


Fig. 1. Illustration of the four "V's" of big data.

2 BACKGROUND ON BIG DATA APPLICATIONS

The "cloud" is a platform that has been used to cost effectively deploy an increasingly wide variety of applications online and make them available to anyone at anytime and anywhere. The rise of cloud computing and cloud data storage, therefore, has facilitated the emergence of big data applications and big data technologies. By providing scalable solutions for processing large volumes of data, big data technologies open up new opportunities to learn insights from data to address the questions that were previously not possible to answer. Big data applications are characterized by four critical features, referred as the four "V's"; Volume, Velocity, Variety, and Veracity, shown in the Figure 1.

- **Volume** refers to the amount of data. Big data is inherently large in volume that must be analyzed and managed to make decisions.
- **Velocity** refers to how fast the data is being received 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. The type of content—structured or unstructured—is also essential for data analysis.
- **Veracity** refers to the level of trust, consistency, and completeness of data. Accumulation of large amounts of data does not guarantee meaningful and accurate data.

Traditionally, cloud servers mainly rely on high-performance CPU cores such as Xeon for processing big data. 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 article are representative programs from 15 different domains such as graph mining, data mining, data analysis platform, and pattern searching

Table 1. Studied 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		Wikipedia	GraphBuilder 0.0.1, Hadoop 1.2.1
	Collaborative Filtering		MovieLens	Hadoop 1.2.1, Mahout 0.6
	Clustering		UCI Machine-learning Repository	
	FP-Growth		Frequent Itemset Mining Dataset Repository	
	SPMF Eclate, RuleGrowth, GSP, SPADE			SPMF 0.96
scale-out	Classification		Wikipedia	Hadoop 1.2.1, Mahout 0.6, Memcached server/client 1.4.15, CloudSuite 2.0
	Graph-Anlysis		Twitter data set	
	Data-caching		Twitter data set	
Traditional	Spec2006		Reference input	Spec 2006
	Parsec		Native input	Parsec 2.1

applications, which are frequently used in real-world applications. There are several recent works that have included these selected applications for benchmarking and characterization of Hadoop-based big data application (Veiga et al. 2016; Hwang et al. 2016; Gao et al. 2013; Dimitrov et al. 2013). We provide these selected applications, along with their particular domain and data type in Table 1. In addition, big data frameworks are evolving continuously and rapidly. New big data frameworks have emerged in recent years in response to continuous changes in the big data analytics field; examples include Spark, Tez, Flink, and Petuum, just to name a few. However, Hadoop MapReduce is still a de facto standard of big data that has been adapted widely by industry and has a large community. While all emerging big data analytics frameworks cannot be included for characterization and benchmarking in one research work, this article mostly focuses on the fundamental Hadoop MapReduce framework. Our future work will investigate the choice of big vs. little core for emerging big data frameworks such as Tez, Flink, Petuum, and Spark.

3.1 Big Data Workload

3.1.1 Hadoop Micro-benchmark. 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 particular words appear in a set of files. Mapper proceeds with a line as an input and fragments it into words. Reducer sums the counts for each word and provides the aggregated value. WordCount *(WC)* is a compute bound application as it performs string processing and generates a very small output (Huang et al. 2010).
- **Sort** uses the MapReduce framework to sort the input directory in the output directory. A mapper is an identity function and directs input records to the appropriate reducer. The

actual sorting occurs in the internal shuffle and sort phase of MapReduce. Finally, the data is transferred to reducer that is also an identity function. Sort *(ST)* micro-benchmark is I/O bound, because it performs very little processing on a large input data and produces a large output data (Huang et al. 2010).

- **Grep** benchmark extracts matching strings provided by the user against text files. It then sorts matching strings by their frequency and stores the output in a single output file. Grep *(GP)* is a compute intensive application. The output size depends on how often the matching string appears that can range from zero to the size of the input.
- **TeraSort** performs a scalable MapReduce-based sort of input data. It first samples the input data and computes the input distribution by calculating the quantiles equal to the number of reducers. This distribution is then used as a partition function that uses a sorted list of N-1 sampled keys that define the key range for each reducer. TeraGen command generates a large random data for the TeraSort *(TS)*. TeraSort is a compute bound in some phases of execution and I/O bound in other phases (Huang et al. 2010). It is mainly used for testing the CPU/Memory power of a node (Huang et al. 2010).
- **TestDFSIO** is a storage throughput test that analyzes how HDFS handles a large number of tasks operating writes or reads concurrently. Each map task opens an HDFS file to write or read sequentially. The aim is to measure the data size and execution time of the task. Finally, the single reduce task aggregates the results of all the maps to calculate the throughput. TestDFSIO is divided into two parts, TestDFSIO-Write and TestDFSIO-Read to write and read data to/from HDFS, respectively (Huang et al. 2010).

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 *(PR)* and *LDA* algorithms implemented on PowerGraph (Willke et al. 2012).

3.1.3 Collaborative Filtering (CF) Recommendation. Apache Mahout implements a widely used collaborative filtering framework that helps recommending items to the users based on their preferences. It uses historical data on user behaviors such as clicks, ratings, views, and purchases to provide recommendations. Collaborative Filtering Recommendation is a supervised machine-learning algorithm that is used by many recommender systems to predict the performance of the users based on their history. The algorithm is feed with the raw input data to make a decision and provide preferences estimation as output. The algorithm learns from the users to better understand their needs. In collaborative filtering, for each item or user, a neighborhood is formed with similar related items or users. Once you make an action to item, the recommendations are drawn from that neighborhood. It suggests the functionality of top-N recommendation items for a specific user by looking the nearest neighbors based on resemblances. Users providing the same rating of the items become the nearest neighbors of others (Collaborative 2017). Database is obtained from the MovieLens website and consists of 100,000 ratings from 943 users on 1,682 movies (Frequent 2015).

3.1.4 Clustering (Clus). Clustering *(Clus)* a fundamental task in Data Mining, 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 (Mahout 2017).

3.1.5 Association Rule Mining. Association Rule Mining is a well-known approach for exploring association between various parameters in large databases. We analyzed Frequent Pattern

(FP)-Growth, a resource intensive program that aims to determine item sets in a group and identifies which items typically appear together (Mahout 2017; Li et al. 2008).

3.1.6 Sequential Pattern Mining Framework (SPMF). 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 (SPMF 2016), for which we have selected Equivalence Class Transformation *(Eclat)*, RuleGrowth *(Rule)*, Generalized Sequential Pattern *(GSP)*, and Sequential Pattern Discovery using Equivalence classes *(SPADE)*.

3.2 Scale-Out Workloads

3.2.1 Classification. Classification technique learns from the existing categorizations and groups the unclassified items to the best corresponding category (Ferdman et al. 2012).

3.2.2 Graph-analysis. Graph-analysis is performed by implementing the TrunkRank on GraphLab (Ferdman et al. 2012). 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 (Ferdman et al. 2012).

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.

While SPEC 2007 and PARSEC benchmarks are old, however they are still important benchmark suites to evaluate and optimize processor architecture. SPEC and PARSEC are widely used benchmark suites for evaluating general purpose CPUs and multi-core architectures. Several prior researches including (Jia et al. 2016; Veiga et al. 2016; Hwang et al. 2016) to understand the behavior of emerging applications (such as scale-out or big data) have also characterized traditional CPU and parallel application for a baseline comparison.

The focus of this article is not on SPEC and PARSEC benchmark, rather it is on big data applications. However, as processor architectures are also optimized for traditional benchmarks such as SPEC and PARSEC, it is important to find out how the performance and power trade-offs between Xeon and Atom changes for big data applications compared to traditional CPU intensive as well as parallel applications.

4 MEASUREMENT TOOLS AND METHODOLOGY

4.1 Hardware/Software Infrastructure

Figure 2 presents a methodology that we used in this study. We conduct our study on two state-of-the-art servers; a three-node Intel Xeon server and a three-node Intel Atom server. Intel Xeon E5-2420 encloses six aggressive processor cores per socket (12 cores per node) with three-level of 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. It is important to note the architectural difference between the two servers, in particular the number of cores, cache hierarchy and even memory subsystems. The goal of this work is not to make a micro-architectural comparison between the two cores, rather is to understand the system and micro-architectural

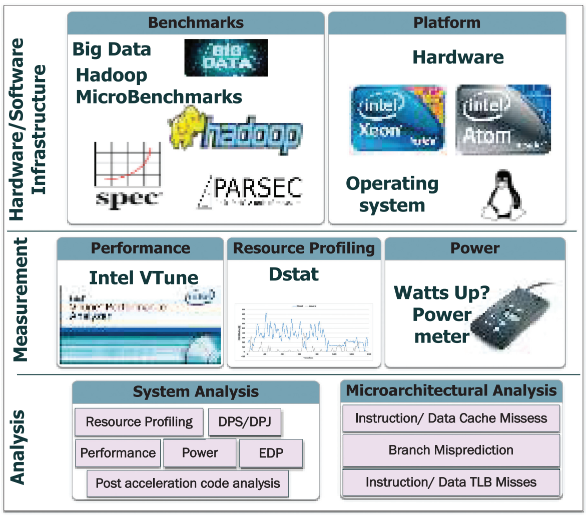


Fig. 2. Methodology.

Table. 2. Architectural Parameters

Processor	**Intel Atom C2758**	**Intel Xeon E5-2420**
Operating Frequency	2.4GHz	1.9GHz
Micro-architecture	Silvermont	Sandy Bridge
Number of Cores	8	12
Hyper-threading	No	No
L1i Cache	32KB	32KB
L1d Cache	24KB	32KB
L2 Cache	4 × 1,024KB	256KB
L3 Cache	—	15MB
System Memory	8GB	32GB
Memory Type	DDR3 1,600MHz	DDR3 1,600MHz

level behavior of the two server classes when running big data applications. This will also help us to find out which server class, whether a low-power Atom-based or a high-performance Xeon-based, suites for big data. The operating system used is Ubuntu 14.10 with Linux kernel 4.11. While network overhead, in general, influences the performance of the studied application for the big data application, Liang et al. (2014) show that the network optimization can only reduce job completion time by the 2%. The network is not a bottleneck, because little data is transmitted through the network. Not to mention Hadoop load the input data according to the locality and combine the intermediate data in the local tasks to packet of size KB before transmission. Like discussed in Ousterhout et al. (2015), network overhead is irrelevant to the overall performance of the application under test even on 1Gbps networks. We have 10Gbit Ethernet connection between nodes, which is significantly higher than applications bandwidth requirements, thus the network did not become the bottleneck for the studied applications.

4.2 Input Data Size

While all experimental results in this article were performed on three-node Atom and three-node Xeon servers, the collected statistics including micro-architectural data as well as system parameters are collected from a single node to mainly compare the big vs. little servers at system and architecture level. Although Hadoop based applications exploits cluster-level infrastructure with many nodes for processing big data applications, the focus of this article is on the system and micro-architecture level analysis per node, to understand the choice of big- vs. little-core servers as a more efficient architecture. For instance, 10GB input data size per node presents 30GB input data size processed by application in a three-node cluster. While MapReduce was originally introduced to process multi-terabyte data in scale-out clusters, most MapReduce workloads have a footprint in the GB range with the median of 14GB (Appuswamy et al. 2013). Hadoop exploits cluster-level infrastructure with many nodes for processing big data applications, however, the experimental data should be collected at the node level to characterize the big data application on high-performance server and low-power server architectures.

4.3 Measurement Tools

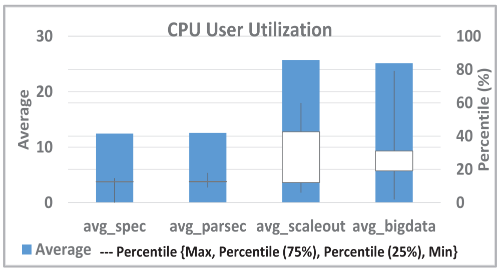
We analyze the architectural behavior using Intel VTune (Intel VTune 2015), a performance-profiling tool that provides an interface to the processor performance counters. We have used Watts up PRO power meter to measure the power consumption of the servers (WattsUpPro 2015). Wattsup power meter produces the power consumption profile every one second. 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 as well as DRAM power to estimate the power consumption of the CPU including the core and on-chip cache subsystem (Blem et al 2013). We also used Dstat (2018), which is a system monitoring tool to collect various statistics of the system at run-time.

4.4 Application Classification

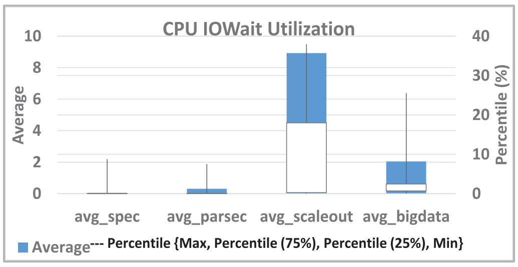
We present the real-time system resources profiling and averaging results to understand the run-time behavior and resource utilizations of Hadoop applications. The recorded metrics are CPU utilization, iowait CPU utilization, Disk I/O bandwidth, and Memory Footprint. CPU utilization reports the load handled by the CPU. Iowait represents the percentage of time CPU is idle while waiting for I/O operation to be completed. Disk I/O bandwidth reports the disk I/O bandwidth rate and Memory Footprint indicates the minimum amount of memory (KB) required to run the application. Numerous studies (Fan et al. 2014; Makrani et al. 2017) have used Dstat tool to understand the runtime status and the resource consumption of the applications under test on the studies servers. Therefore, we have used resource utilization features as well as the micro-architectural parameters to classifies the application into one of the three classes, i.e., Compute-bound (C), Hybrid (H) and I/O-bound (I). Micro-architectural parameters include IPC, Instruction Cache Misses per Kilo instructions (MPKI), LLC cache MPKI, and Branch Misprediction rate. The higher LLC cache MPKI translates into the number of memory access. Thus, the application with the low CPU utilization, iowait CPU utilization, high memory footprints, and high LLC MPKI readings classifies the application as memory bound application.

5 EXPERIMENTAL RESULTS AND ANALYSIS

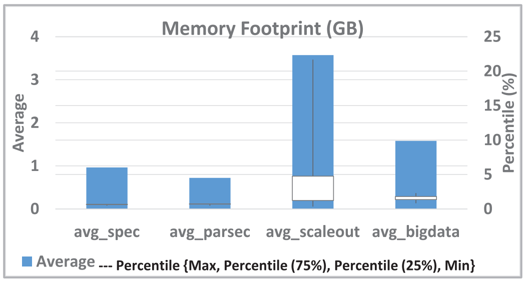
In this section, we discuss application-level, system-level and micro-architecture-level analysis results, when running traditional CPU benchmarks, parallel benchmarks, scale-out, and big data



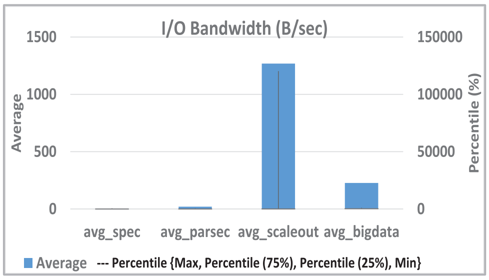
(a): CPU User Utilization



(b) CPU iowait Utilization



(c): Memory Footprints



(d) I/O Bandwidth

Fig. 3. Resource utilization analysis of all the studied workloads on Atom.

applications. We only report the average results for SPEC, PARSEC, and Scale-Out applications and compare them with the big data applications.

5.1 System-Level analysis

In this section, we perform system-level analysis of big data applications on big and little core-based servers. We explore the resource utilization of big data applications by examining the CPU user utilization, CPU iowait utilization, Disk I/O bandwidth (read/write) rate and memory footprints at coarse-grained and fine-grained levels. Based on the resource utilization features and micro-architectural parameters, we implement the code that classifies the application into one of the three classes, i.e., Compute-bound (C), Hybrid (H), and I/O-bound (I).

5.1.1 Workloads Resource Utilization Analysis. Figures 3(a)–3(d) shows the percentile and average utilization results for traditional CPU, parallel benchmarks, scale-out as well as big data applications on little core. SPEC 2006 and PARSEC are two widely used benchmark suites for evaluating general purpose CPUs and multi-core systems. Figures 3(a) and 3(b) show that, on average, the CPU user utilization of SPEC2006 and PARSEC Benchmarks suite is 12.46% and 12.57%, and their CPU iowait utilization is 0.046% and 0.309%, respectively. Memory footprint illustrates the memory usage in the range of 700MB to 1.1GB where an application with more than 1GB is considered as memory intensive as shown in Figure 3(c). Figure 3(d) shows the total disk bandwidth (read and write) of big data applications along with the average results for traditional, parallel and scale-out applications. Scale-out applications have the highest disk utilization. Moreover, a number of big data applications have heavy disk utilization. However, SPEC2006 and PARSEC are not I/O bound applications. We observe that unlike traditional CPU and parallel benchmarks, Scale-out and big data applications exhibit hybrid behavior of a combination of high computational, memory, I/O, and control intensities. (Control intensity details are presented in Section 5.6). It is important to note that most of big data applications studied in this work rely on Hadoop MapReduce

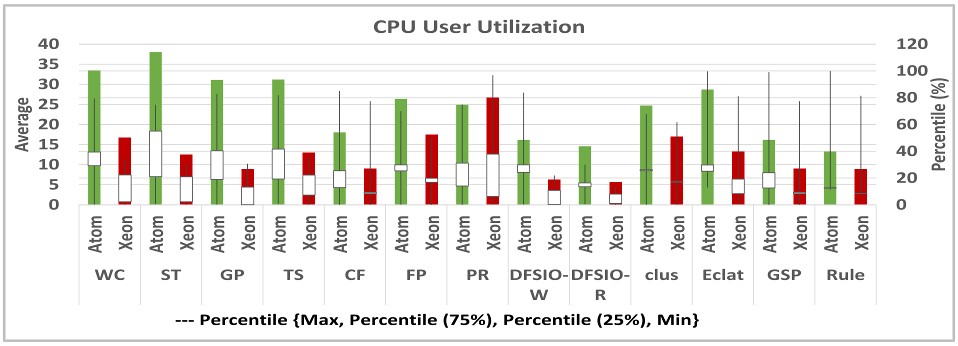


Fig. 4(a). Average and Percentile CPU user utilization for Big Data applications on Big and Little core.

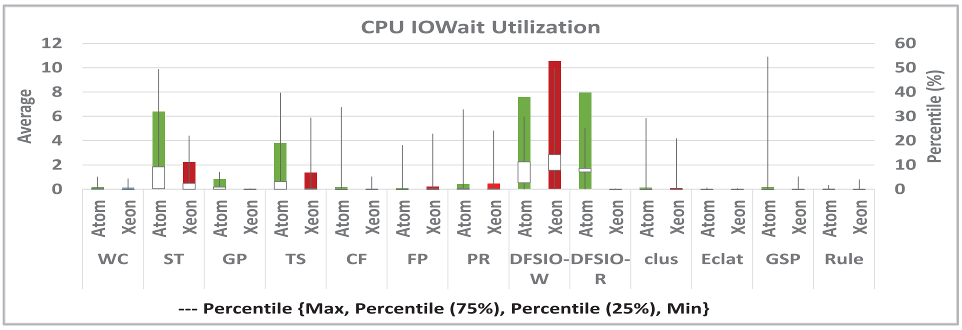


Fig. 4(b). Average and Percentile CPU iowait utilization for Big Data applications on Big and Little core.

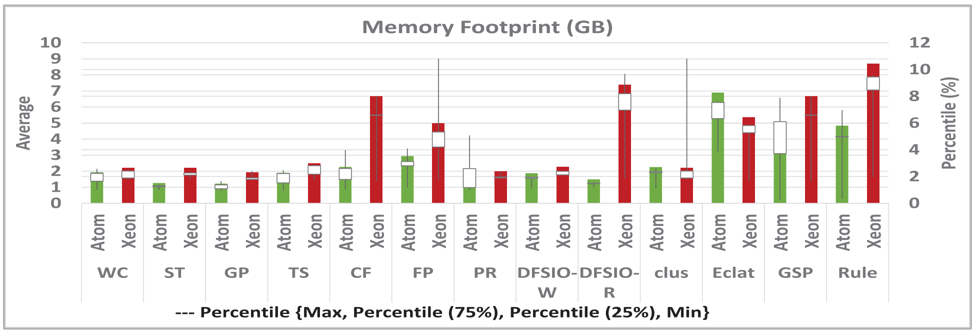


Fig. 4(c). Average and Percentile Memory Footprint for Big Data applications on Big and Little core.

framework. Considering that most Hadoop based applications contains more than one MapReduce job, many of them are shown to have hybrid behavior; a combination of compute intensive, memory intensive, control intensive, or I/O-intensive.

5.1.2 Resource Utilization Analysis of Big Data Applications on Big and Little Core. Figures 4(a)–4(e) show the percentile and average resource utilization analysis of big data applications on big and little core. Figures 4(a) and 4(b) show that across all studied big data applications the average CPU utilization—user and iowait—is approximately 2× higher on Atom as compared to Xeon. The main reason is the low processing capability of Atom core that can process only two instructions

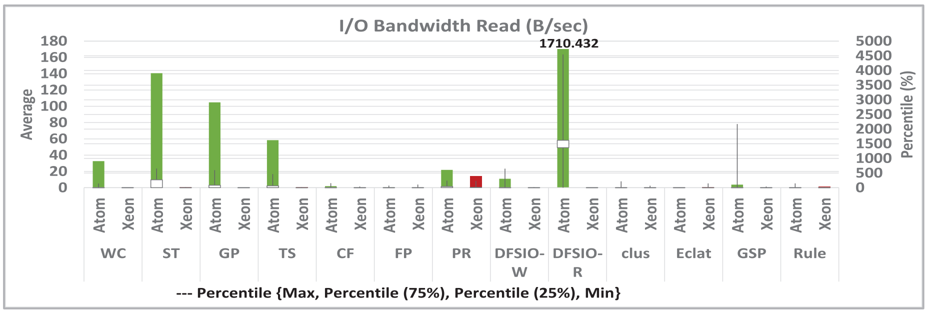


Fig. 4(d). Average and Percentile I/O Read Bandwidth for Big Data applications on Big and Little core.

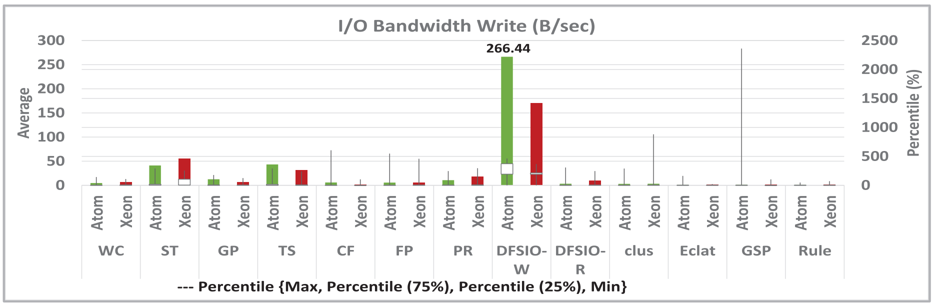


Fig. 4(e). Average and Percentile I/O Write Bandwidth for Big Data applications on Big and Little core.

simultaneously while Xeon can process four instructions per cycle. The results show that the CPU user utilization gap between Atom and Xeon is higher for Sort, DFSIO-write, Grep, and TeraSort (26%, 20%, 22%, and 18%, respectively) as compared to other applications that have CPU user utilization gap lower than 15%. This trend can be used to classify applications differently on both servers (explained later in Section 5.1.3).

Figure 4(c) shows higher memory footprints on Xeon as compared to Atom. Considering Xeon has a large memory compared to Atom, clearly the performance of the big data applications improves on Xeon over Atom; however, the important question is whether Atom can satisfy the memory requirements to process the big data applications. To address this question, we will discuss the detail micro-architectural analysis in Section 5.6. Figures 4(d) and 4(e) show that on average the I/O bandwidth (read/write) rate is higher on Atom compared to Xeon across all the studied big data applications, especially in Sort, Grep, DFSIO-write, DFSIO-read applications. Moreover, the applications with a higher CPU user utilization (Wordcount, PR, and TeraSort) has noticeably higher I/O read/write rates on Atom compared to Xeon. This behavior can lead the applications to be classified differently on Atom and Xeon.

5.1.3 Application Classification Analysis. Based on the average resource utilization analysis, we have observed that big data applications illustrate different behavior on Atom and Xeon servers. This necessitates the application to be classified on both Atom and Xeon to understand the runtime behavior and the resource utilization. In addition, we have selected one application from each classification category to illustrate the run-time behavior of big data application on big and little core servers as shown in Figure 5.

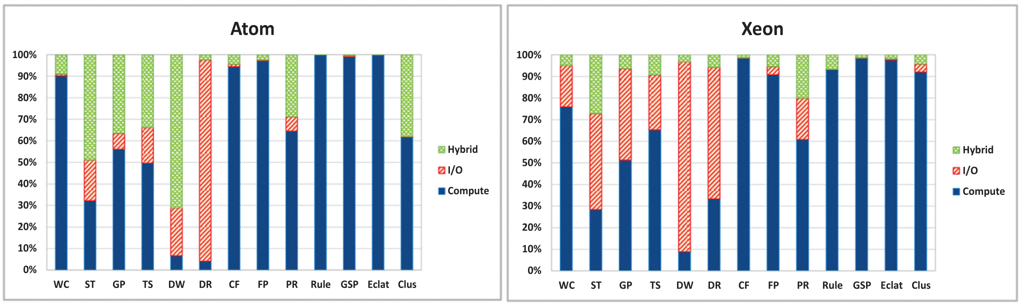


Fig. 5. Fine-grained analysis of big data applications on (a) little core and (b) big core.

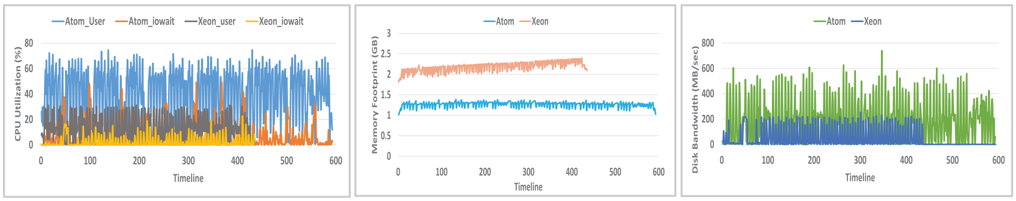


Fig. 6. Runtime [1s interval]. (a) CPU and iowait utilization. (b) Memory Footprint. (c) Disk bandwidth of Sort.

5.1.3.1 Coarse-grained Level. To determine the coarse-grained level behavior, we have evaluated the average resource utilization results of the big data applications on big and little core, as shown in Figure 4. The CPU user utilization is higher than the average user utilization in Wordcount, FP, PR, Clus, and Eclat (16.32%, 17.4%, 26.6%, 17.01%, and 13.28%, respectively), with their low CPU iowait utilization and I/O bandwidth (read/write) rates, categorizing these applications as compute intensive on Xeon. The CPU user utilization of these applications is higher on Atom compared to Xeon; therefore, they illustrate a similar behavior on Atom. Although, a higher CPU user utilization for TeraSort on Xeon (13.05%) and Atom (31.2%) makes it comparable to other compute bound applications, the high iowait utilization (3.78%) and moderate I/O bandwidth (read/write) rates especially on Atom illustrates a hybrid behavior. The basic principle of TeraSort benchmark is similar to Sort; however, unlike Sort, TeraSort has a reduce phase to improve the performance (detailed analysis is presented in Section 5.1.3.2).

Sort, DFSIO-Read, and DFSIO-Write have low CPU user utilization (12%, 6.4%, and 5.6%, respectively) with high iowait utilization and IO bandwidth (read/write) rates on Xeon. This behavior classifies these applications as I/O bound. Similar trend has been observed for the DFSIO-Write and DFSIO-Read on Atom. The CPU user utilization (16.6% and 14.57% for DFSIO-Write and DFSIO-Read, respectively) is low compared to other applications on Atom while the CPU iowait utilization (7.59% and 7.96%, respectively) is very high. In contrast, the CPU user utilization for Sort is very high (38.02%) as well as the CPU iowait utilization (6.38%). This behavior is consistent with Figure 4(a) percentile results that illustrate large variations in the CPU utilization, ranging from 20% to 55%. Additionally, the disk I/O-read/write readings are noticeably high in sort benchmark as shown in Figure 6. Consequently, sort benchmark is classified differently on Atom as hybrid, than Xeon as I/O bound. Grep, CF, GSP, and Rule have relatively low CPU user utilization as well as CPU iowait utilization that is near zero. In addition, their I/O bandwidth (read/write) rate is very low. Similar trend has been observed for these applications on Xeon with the exception of Grep,

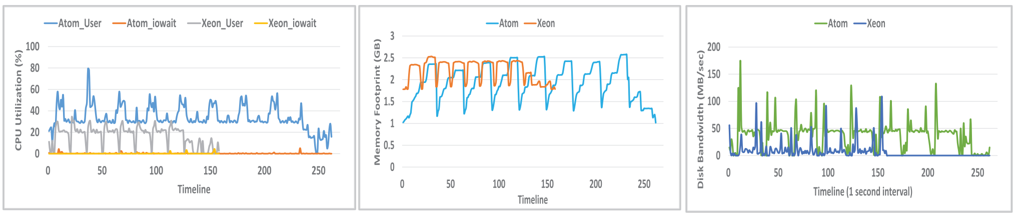


Fig. 7. Runtime [1s interval]. (a) CPU and iowait utilization. (b) Memory Footprint. (c) Disk bandwidth of Wordcount.

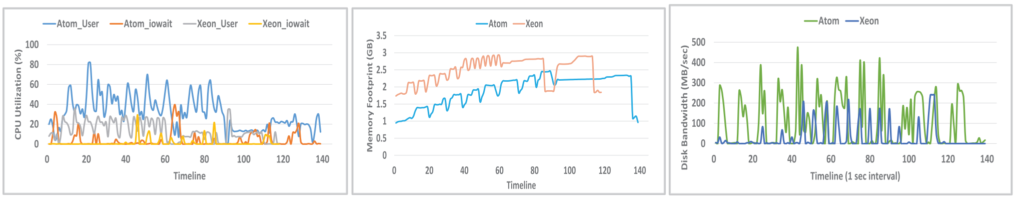


Fig. 8. Runtime [1s interval]. (a) CPU and iowait utilization. (b) Memory Footprint. (c) Disk bandwidth of TeraSort.

which has very high I/O bandwidth (read/write) rate on Atom. To better understand their behavior, we have evaluated these applications at the fine-grained level to classify their phases at the run-time.

5.1.3.2 Fine-grained Level. To evaluate application behavior effectively, we classify applications as compute, I/O or hybrid tasks at a fine-grained level of one second. Figures 5(a) and 5(b) show the accumulated fraction of compute, I/O, and hybrid tasks out of entire application execution on both Atom and Xeon. Compute bound applications' behavior at a fine-grained level is consistent to what we have observed at a coarse-grained level. Wordcount, FP, PR, Eclat, and Clus are dominated by a significantly large fraction of compute bound tasks (on average 70% of the total application is computationally intensive). We have selected Wordcount to present the run-time behavior. For Atom, the CPU user utilization is shown to be high (33.4%) with no significant variation except few spikes, as shown in the Figure 7(a) while the iowait is very close to zero (0.19%). Similar trend is observed on Xeon with lower CPU user utilization and I/O write/read rate as shown in Figure 7.

On the other hand, TeraSort contains slightly larger fraction of computationally intensive tasks as compared to I/O hybrid tasks that can construe the entire application as compute bound application. However, on Atom, the application behavior can be interpreted as hybrid. Figure 8 shows the run-time behavior of TeraSort on Atom and Xeon. We observe that Atom and Xeon CPU user utilization difference in the map phase is high, however, the difference among the two is small in the reduce phase. Figure 8(c) shows heavy but unstable I/O-read/write rate on Atom and significantly lower I/O-read/write rate on Xeon. TeraSort have heavy CPU utilization in the map phase heavy I/O-write rate during the reduce phases. Considering Map phase is computational intensive and with longer execution time than reduce phase, this application is classified as compute bound on Xeon. However, Atom illustrates an opposite trend, which shows the studied application to be I/O bound for almost half of the execution time. Thus, on Atom this application is classified as hybrid.

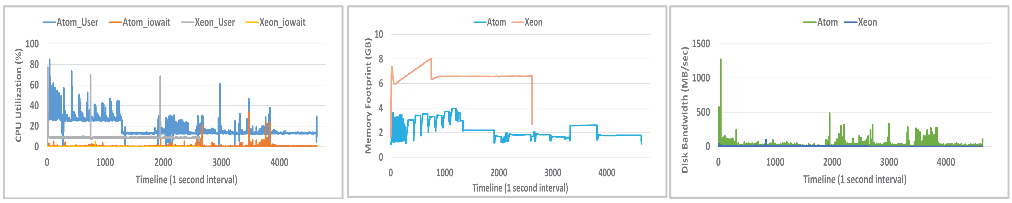


Fig. 9. Runtime [1s interval]. (a) CPU and iowait utilization. (b) Memory Footprint. (c) Disk bandwidth of CF.

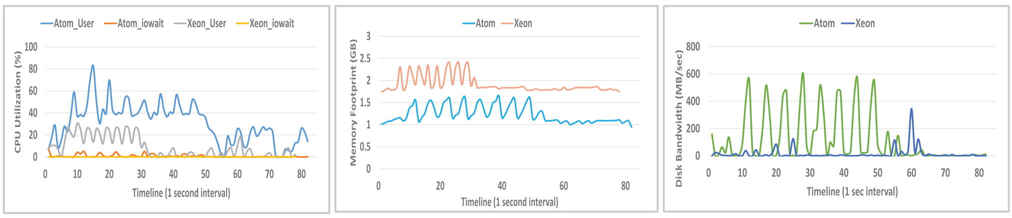


Fig. 10. Runtime [1s interval]. (a) CPU and iowait utilization. (b) Memory Footprint. (c) Disk bandwidth of Grep.

In addition, the behavior of CF, GSP, and Rule can be clearly classified as compute bound applications at the fine-grained level. Major fractions of execution time of these applications are computationally intensive, as shown in Figure 5. Figure 9 presents the runtime behavior of CF. CF is divided into nine different MapReduce jobs such as recommender job, item filtering job, similarity matrix job, and aggregation job (44). When running the CF on Atom, shown in Figure 9(a), the first two MapReduce jobs have heavy CPU utilization that reduces from 35% to 15% for the remaining MapReduce jobs. Xeon illustrates a stable CPU user utilization around 10%, as shown in Figure 9(a). Over the entire interval, Atom on average has 1.43× higher CPU user utilization than Xeon. However, if we observe the MapReduce task with respect to individual MapReduce jobs, then we find that last MapReduce jobs in CF have relatively lower CPU user utilization that is different on Atom compared to Xeon. This behavior indicates a clear advantage to run these phases on little Atom servers, because unlike low-power embedded server Atom, Big-core Xeon servers have high power consumption.

Although, Grep behavior is not clear at coarse-grained level, it illustrates a hybrid behavior at fine-grained level. Grep application contains two MapReduce jobs, the grep-search and grep-sort. The first job is more time-consuming and CPU-intensive as compared to the second job, as shown in Figure 10.

We observe that the behavior of compute-bound applications remains almost the same on Xeon and Atom. However, due to low processing capacity of little Atom, applications that are classified as I/O bound on Xeon illustrate hybrid behavior on Atom. In addition, the CPU user utilization difference between Atom and Xeon is small for reduce phase of the Hadoop applications. Thus, reduce phase clearly favors the little Atom server, considering the effective core utilization and lower power consumption of Atom.

5.2 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. We have conducted the data size sensitivity analysis of Hadoop micro-benchmarks with the dataset of 10MB, 100MB, 1GB, and 10GB per

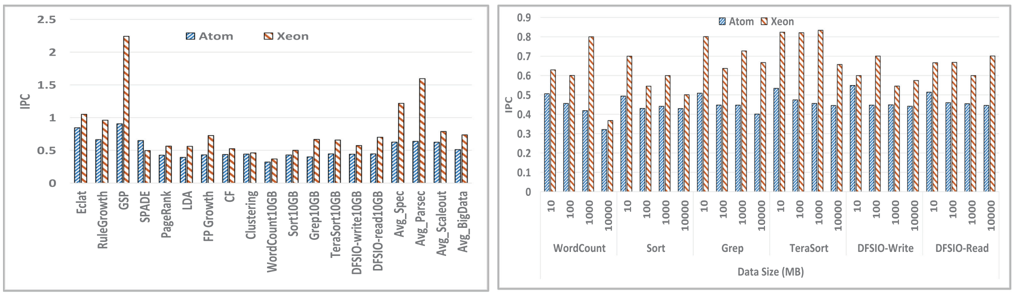


Fig. 11. IPC. (a) Big Data workloads. (b) Different datasets of Hadoop micro-benchmarks.

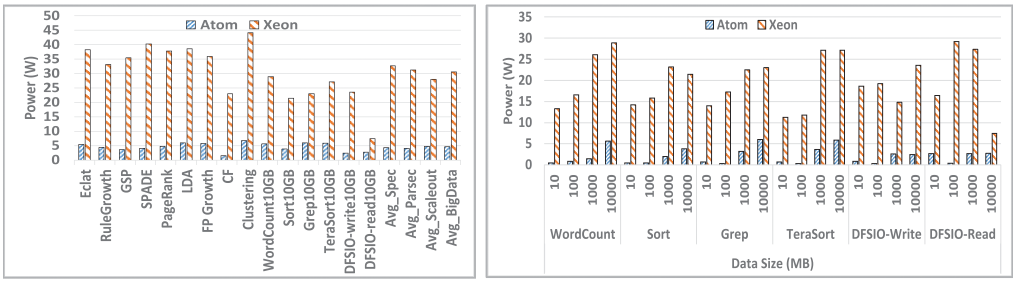


Fig. 12. Power Reading (a) Big Data workloads (b) Different datasets of Hadoop micro-benchmarks.

node to understand the impact of the size of data per processing node on system-level as well as microarchitecture-level parameters. Figure 11(a) presents that the average IPC of big data is 1.65× lower than traditional CPU benchmarks on big core and 1.21× lower 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× 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 11(b) shows the IPC of Hadoop micro-benchmarks for different data sizes. The results are consistent with the results in Figure 11(a) 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 5.6.1). 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 1,000MB 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.

5.3 Power Consumption and Energy-Efficiency Analysis

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

5.3.1 Power Characterization. Figure 12(a) shows the average 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. Big core consumes on average 35 Watts of dynamic power with the

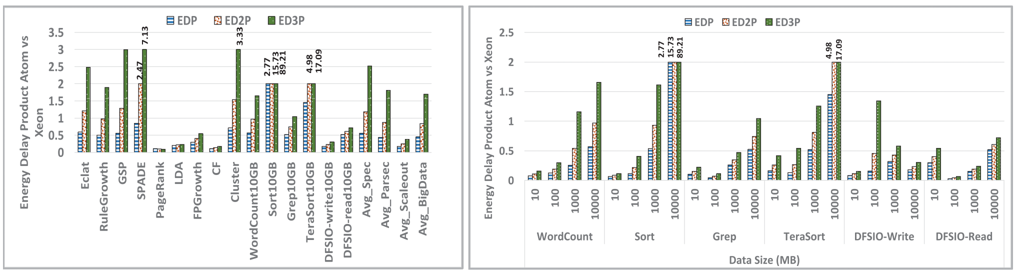


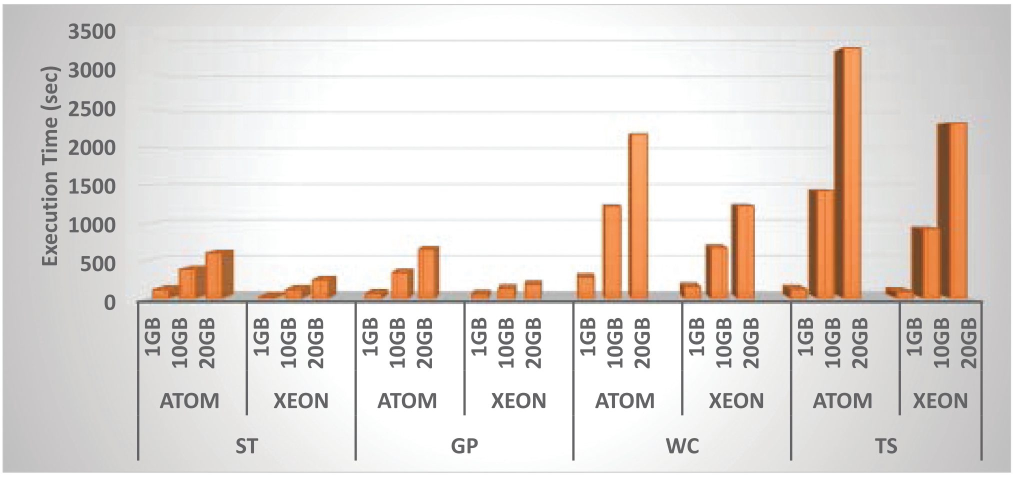
Fig. 13. EDP, ED^{2}P, and ED^{3}P Analysis. (a) Big Data Workloads. (b) Different datasets of Hadoop micro-benchmarks.

peak of 44 Watts (in CF application). Little core consumes much lower power as expected, ranging from 0.9 to 6 Watts with an average of 4.8 Watts. Figure 12(b) shows that the power consumption increases as the size of data per node increases in most cases across both big and little core architectures. This is more noticeable in little core. While increasing the 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 18). Therefore, for low-end little core, where cache, DRAM, and off-chip components are dominant power consumers (unlike high-performance Xeon core), a clear rise in power consumption is observed as size of data increases.

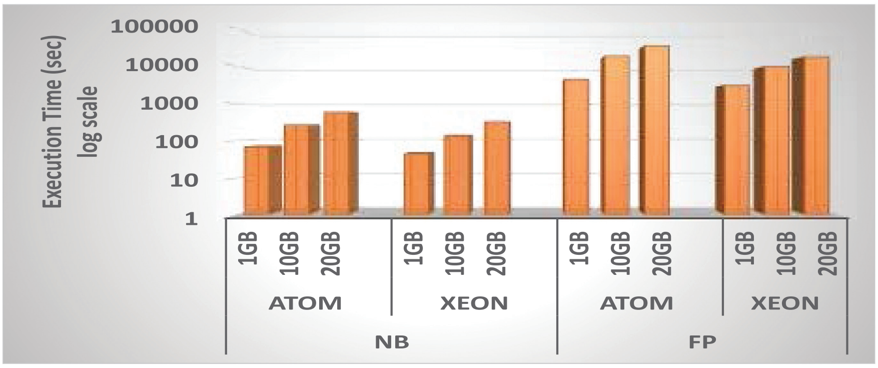
5.3.2 Energy-Efficiency Analysis with near Real-Time Processing Constraints. Based on the results of power consumption for both platforms, we have evaluated the trade-off between power and performance by investigating the EDP metric. Furthermore, we have explored the ED^{2}P and ED^{3}P to understand the impact of near real-time performance constraints on big data applications and how more performance constraints affects the choice of most efficient server architecture. Figure 13(a) illustrates EDP, ED^{2}P, and ED^{3}P ratio for little vs. big cores. The ED^{X}P (X=1,2,3) results show that compared to little core server, big-core server is noticeably more efficient for traditional CPU applications in comparison with big data. Also, interestingly for scale-out benchmarks, little core is always more efficient than big core for EDP, ED^{2}P, and ED^{3}P 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 Sort and TeraSort, for large data sizes (10GB), 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 (2× 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^{2}P and ED^{3}P metrics, big core becomes more efficient compared to little cores across most applications. Figure 13(b) 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 smaller data sizes. However, as the size of data increases and with more performance constraints big-core server becomes more efficient.

5.3.3 Input Data Size Sensitivity Analysis. In this section, we study the impact of input data size on performance and energy-efficiency of Atom vs. Xeon. We have conducted the data size sensitivity analysis of Hadoop micro-benchmarks and selected real-world applications (NB and



(a): Hadoop micro-benchmarks



(b) Real world applications

Fig. 14. Execution time of big data applications at various input data size.

FP) with the datasets of 1GB, 10GB, and 20GB per node. For this research, we have used input data sizes per node for each application. For instance, 20GB input data size per node presents 60GB input data size processed by application in a three-node cluster.

As shown in the Figures 14(a) and 14(b), the execution time is proportional to the input data size. Comparing the two architectures, we can observe that the execution time increases significantly more on Atom as a function of data size (10.15×, 7.75×, 27.15×, 8.59×, and 7.97×) compared to Xeon (3.45×, 7.75×, 26.07×, 7.22×, and 5.96×) for GP, WC, TS, NB, and FP, respectively, as shown in Figures 14(a) and 14(b). This is mainly due to the performance bottleneck of Atom core that exist in its compute capacity as well as memory subsystem, which is exposed more as the size of data increases.

To evaluate the trade-off of power and performance between Xeon and Atom, we have investigated the EDP metric (Figure 15). We have observed a clear trend of rise in EDP with the increase in input data size across both architectures. Across almost all the studied benchmarks little Atom core is clearly a favorite choice for energy-efficiency, with the exception of the Sort application. The increase in the data size progressively makes the big core more efficient compared to the little

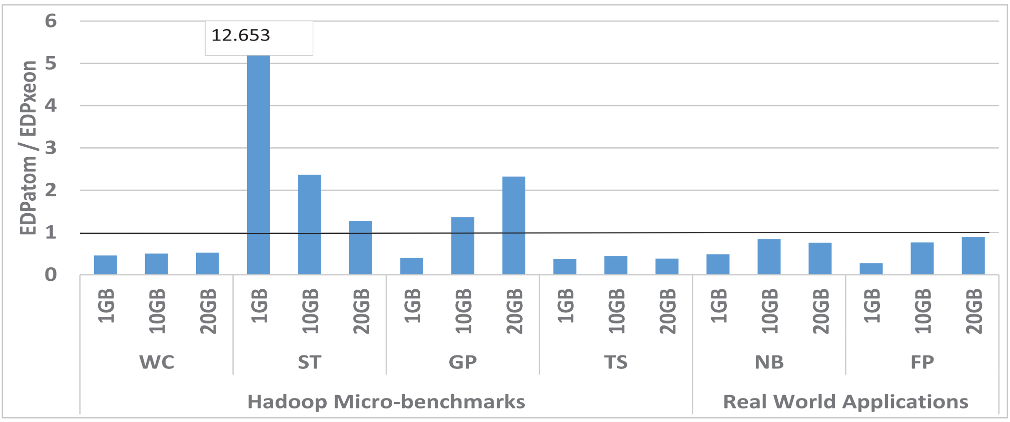


Fig. 15. EDP analysis of big data applications with various input data size.

core across all the applications with the exception of Sort that illustrate the opposite trend. For the smaller data size, the higher processing capability of big core hides the I/O communication cost effectively. However, the rise in data size aggravates the I/O cost to an extent that it diminishes the benefit of the high processing capability of the big core for I/O intensive benchmarks. In sum, the studied applications showing clear benefit for little core, except for I/O bound applications and also when the size of data per node increases.

5.4 Dps-Dpj Analysis

In this section, we evaluate the data processing capability of big and little core-based servers for various data sizes in Hadoop micro-benchmarks. The results are shown in Figures 16(a) and 16(b). For most applications, on both big- and little-core-based servers, 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 1GB of size, while in other applications it occurs in at least an order of magnitude larger data size. The reason is that the rise in data size exacerbates the I/O cost to an extent that it diminishes the benefit of high processing capacity in hybrid and I/O intensive applications. The DPS difference between big- and little-core-based servers is becoming larger for CPU intensive applications such as word-count 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 big and little servers 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 insignificant. For CPU intensive applications including wordcount 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 Xeon-based server. However, in CPU intensive micro-benchmarks such as WordCount, the DPJ gap between big and little core-based server

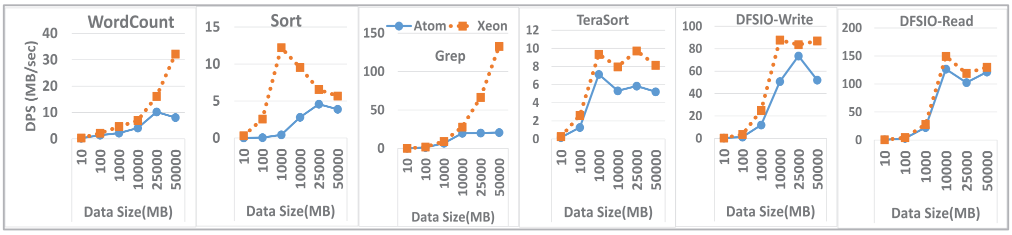


Fig. 16(a). DPS analysis of different datasets of Hadoop micro-benchmarks.

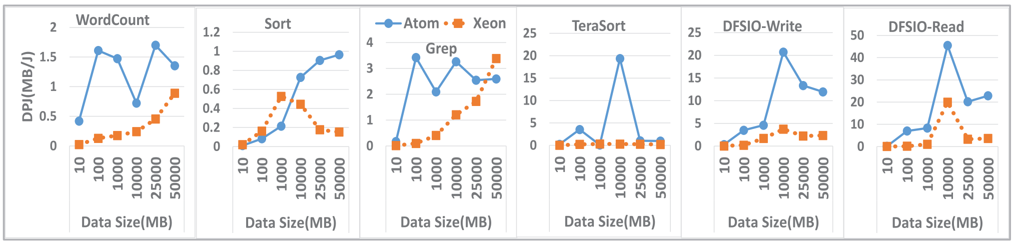


Fig. 16(b). DPJ analysis of different datasets of Hadoop micro-benchmarks.

reduces with the increase in data size. It is also interesting to observe that DPJ of Xeon can exceed Atom in a number of applications and across several different data sizes.

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

5.5 Performance Hotspot and Post-Acceleration CPU Code Characterization

In recent years, the interest in hardware acceleration has revived mainly due to dark silicon challenge. As chips are hitting power limits, computing systems are moving away from general-purpose designs and toward greater specialization. In addition to big, medium, and small cores, the integration of domain-specific accelerators, such as GPUs and FPGAs has been emerging in data center architectures. To find out the right server architecture for big data processing, it is important to understand how deploying an accelerator would necessitate adapting the choice of CPU. For the post-acceleration code characteristics, 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. While the GPU-based platforms have achieved significant speedup across a wide range of benchmarks, their high power demands preclude them for energy-efficient computing (Stolz et al. 2010). FPGAs have shown to be more energy efficient (Fowers et al. 2012), and they allow the exploitation of fine-grained parallelism in the algorithms. Moreover, advances in the FPGA technology advances, suggest that new generation of FPGAs will outperform GPUs. In Nurvitadhi et al. (2017) the authors compare two generations of Intel FPGAs (Arria 10, Stratix10) against the latest highest performance Titan X Pascal GPU. For a ResNet case study, their results show that for Ternary ResNet (Kundu et al. 2017), the Stratix 10 FPGA can deliver 60% better performance over Titan X Pascal GPU, while being 2.3× better in performance/watt showing that FPGAs may become the platform of choice for accelerating the applications. Therefore, we have considered FPGA as a domain-specific accelerator.

A key research challenge for heterogeneous architecture that integrates CPU and accelerator such as FPGA is workload partitioning and mapping of a given application (also referred as

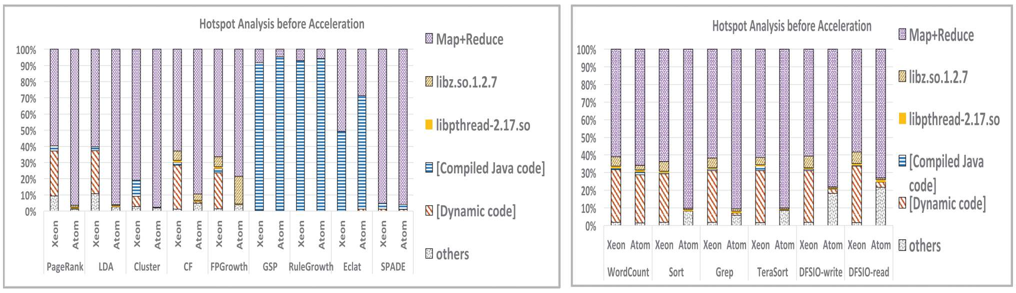


Fig. 17. Hotspot Analysis before acceleration. (a) Big data Applications. (b) Different datasets of Hadoop micro-benchmarks.

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 regions. These regions are candidates for FPGA acceleration, as long as the overhead of communication with the CPU is not significant (Nilakantan et al. 2013). To perform hotspot analysis on big data applications, we use Intel vTune to select the common hotspot modules of an application running on big and little cores. First, we identify and analyze hotspot modules based on their execution time. Figure 17(a) shows the common hotspot modules of studied big data applications and Figure 17(b) 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 for grep and sort. 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 2×. 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. The hotspot modules are similar on a big and little core in most of the studied cases; however, their percentage can vary. Although some applications can illustrate hotspot modules that were dissimilar on both architecture, their impact was negligible. Also, these results demonstrate that Map and Reduce tasks represent the hot spot computation part of the application. Several recent work proposed offloading these hotspot functions, mainly mapper and reducer, to an FPGA and showed significant performance improvement (Accelerating 2013; Kukunas et al. 2014; Khavari et al. 2014). To analyze post-accelerated code and the choice of server architecture, we assume map and reduce tasks are offloaded to an accelerator (Shan et al. 2010; Honjo et al. 2013; Lin et al. 2013). Based on this observation, we have offloaded the map and reduced tasks to the accelerator to understand the choice of server architecture to analyze post-accelerated code that remains on the CPU. The execution time of the hotspot computation part of the application after acceleration consists of three major parts, i.e., time_cpu, the software part of the map phase that remains on the CPU; time_fpga, the hardware part of the map function that is offloaded to the FPGA; and time_trans, the data transfer time between the FPGA and the CPU core. Time_trans is calculated based on the speed of the connections link and the amount

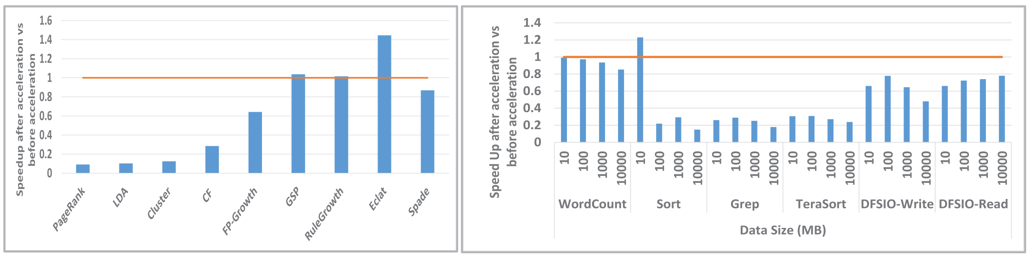


Fig. 18. Speed up of Atom vs. Xeon before and after acceleration. (a) Big data applications. (b) Different datasets of Hadoop micro-benchmarks.

of data that is transferred between the accelerator and the CPU. We do not make any assumption about the speedup gain on accelerator. Note that the speed up gain as a result of acceleration does not change the outcome of our analysis whether big core or little core is best to run the remaining post-accelerated code. To this end, without diving into how each application can be accelerated, we can study the remaining functions that are left for the big or little core-based server to run, where speedup is achieved as time_allCPU/(time_cpu + time_fpga + time_trans), as in Figure 18. After calculating the new execution time, we can study the remaining modules that are left for the big or little core-based server to run.

To determine which server (big or little core server) is best suited to process the code that remains on the CPU after we offload hotspot map and reduce task to an accelerator such as FPGA, we analyze the post-acceleration code. Figure 16 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

$$\boldsymbol{Speedup} = \frac{\left(Exectime_{Atom} / Exectime_{Xeon}\right)_{remaining\ code\ after\ acceleration}}{\left(Exectime_{Atom} / Exectime_{Xeon}\right)_{entire\ application}}. \tag{1}$$

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

$(\text{Exectime}_{\text{Atom}} / \text{Exectime}_{\text{Xeon}})_{\text{entire applications}}$ represents the speed up obtained by migrating the code 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 16 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 in 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 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

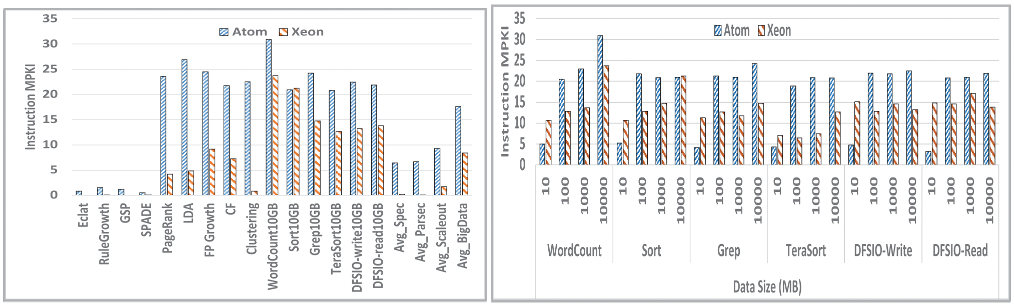


Fig. 19. L1 Instruction MPKI of (a) big data workloads and (b) different datasets of Hadoop micro-benchmarks.

clearly favor little core post acceleration, in some applications post accelerated code show higher speed-up on big core over little core-base server compared to pre-acceleration. Arora et al. (2012) showed that after offloading graphic application to GPU, the remaining code that is left for CPU for processing has different characteristics, as the highly parallel part has offloaded and the control intensive part is left; therefore, their work indicated that for graphic application, we need a high performance, with complex branch predictor for post GPU code. So, initially our prediction was that for post-accelerated big data workload we will observe similar behavior; big core with sophisticated branch predictor is needed: however, our assessment proved this to be different, as for big data applications, for some big core is favored and for others little core is favored. We mostly found that Hadoop-based applications prefer little-core post-acceleration, while others mostly prefer big-core servers.

5.6 Microarchitecture-Level Analysis

To provide insights 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 scale-out applications. Comprehensive microarchitecture analysis helps determine the performance bottlenecks in both big and little core servers when running big data applications. Our work not only discusses the impact of big data applications on high-performance server Xeon, it also compares it with the low-power embedded server to illustrate that compared to traditional applications, which are not energy-efficient for processing on little core, for big data applications it is energy-efficient for processing on little core. The details of the studied parameters are explained in this section.

5.6.1 Instruction Cache Misses. Frequent cache misses decrease the fetch rate and introduce 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 19(a) presents the L1 Instruction cache misses per kilo instructions (MPKI) on big and little core servers. The average I-cache MPKI of big data applications is at least 2.6× higher than the traditional benchmarks and 1.86× higher than the scale-out applications on Atom, while on Xeon it is 3.8× higher than traditional benchmarks and 4.76× higher than scale-out applications. In comparison to Xeon, Atom shows 2× higher I-cache MPKI in big data applications. Figure 19(b) shows data sensitivity analysis of Hadoop micro-benchmarks. Data sensitivity analysis results show an increase in I-cache MPKI on both Xeon and Atom as the size of data increases.

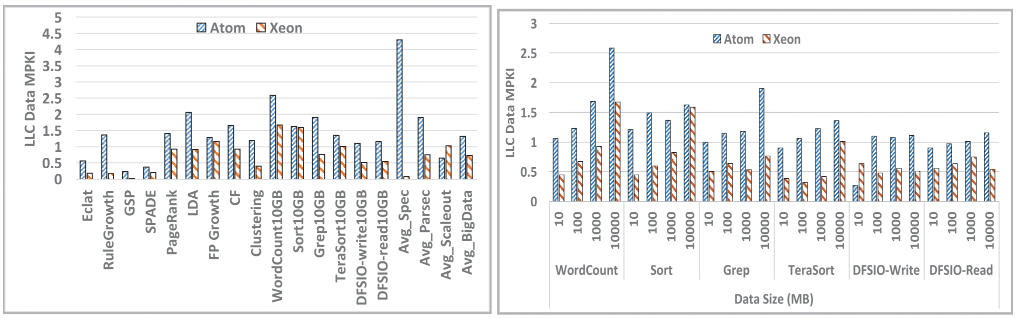


Fig. 20. LLC Data MPKI of (a) big data workloads and (b) different datasets of Hadoop micro-benchmarks.

Observation: The general observation is that the instruction footprints of the big data applications do not fit in the L1 instruction cache on both Xeon and Atom. Similar observation was made in a recent work (Wang et al. 2013) for Xeon. Fast cache access requirement 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 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. Considering big data workloads have deep software stacks, a large number of system calls, the increase in data also increases the dynamic and static linking that results into higher instruction cache misses. Also Increase in data size mostly increases the number of logical mappers (logical mapper = data size / HDFS block size), therefore requiring new set of mappers to be fetch into the cache followed by reduces, therefore increases the cache misses every time new mapper need to be fetch into the cache. This behavior implies that higher I-cache performance is required for big data workloads. Moreover, low-end microarchitecture such as little Atom for big data processing needs to include an instruction prefetchers that can predict complex instruction flow patterns with the advance replacement policy to eliminate the wasted cycles caused by front-end stalls.

5.6.2 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 analyzed the LLC of both processors: L3 cache of Xeon and L2 cache of Atom. Figure 20(a) shows the LLC MPKI on Atom and Xeon for big data applications along with average LLC MPKI of SPEC, PARSEC, and Scale-Out applications. The average LLC data cache MPKI of Big data is at least 2.5× higher than the traditional benchmarks and 3.8× higher than Scale-Out applications on Xeon. In contrast, Atom shows an opposite behavior; on Atom, SPEC applications experiences 3.25× higher data MPKI than big data applications. Moreover, the average LLC data cache MPKI of big data on Atom is 1.8× higher than Xeon. Figure 20(b) 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 high data locality corroborating the observation in Wang et al. (2013) and Ferdman et al. (2012). Furthermore, the results show that while a large three-level cache hierarchy on Xeon has advantage over a two-level cache on Atom, a small 4x1MB LLC cache on 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

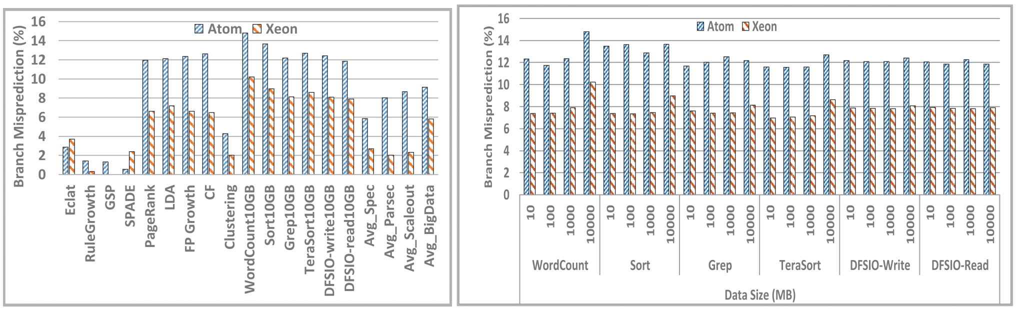


Fig. 21. Branch Misprediction of (a) big data workloads (b) different datasets of Hadoop micro-benchmarks.

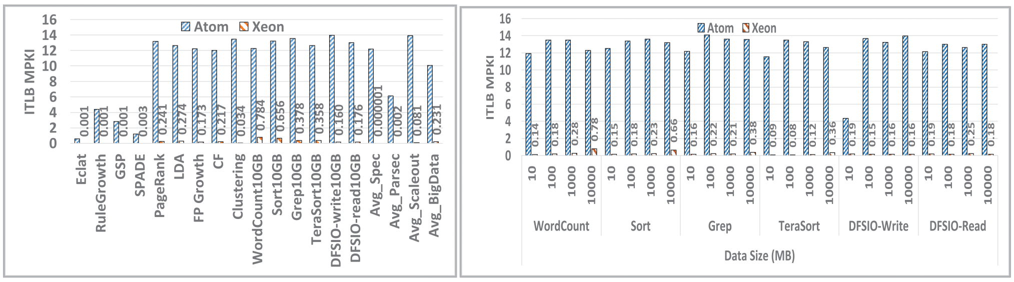


Fig. 22. ITLB MPKI. (a) Big data workloads. (b) Different datasets of Hadoop micro-benchmarks.

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.

5.6.3 Branch Misprediction. Figures 21(a) and 21(b) presents the branch misprediction results. The average misprediction in big data on Atom is 9.3%, while for SPEC, PARSEC, and Scale-Out this is 5.8%, 8.01%, and 8.66%, respectively. In addition, on Xeon, average misprediction of big data is 5.9%, while 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× higher on Atom and 2.13× higher on Xeon compared to traditional benchmarks. In comparison, Atom experiences 1.5× higher branch misprediction than Xeon. Figure 21(b) results show that the branch misprediction rate does not change noticeably with changing the data size.

Observation: Traditional CPU and parallel benchmarks have instruction working sets with tight loops (as in matrix algebra), therefore making them easy for the branch predictor to predict correctly and reduce their misprediction rate. In big data workloads, however, loops are seldom, and instead we have 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.

5.6.4 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 report the Instruction TLB (ITLB) and Data TLB (DTLB) MPKI in Figures 22 and 23. Our results show that on Xeon big data applications have the highest ITLB MPKI with the average of 0.23, while for SPEC, PARSEC, and Scale-Out this 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,

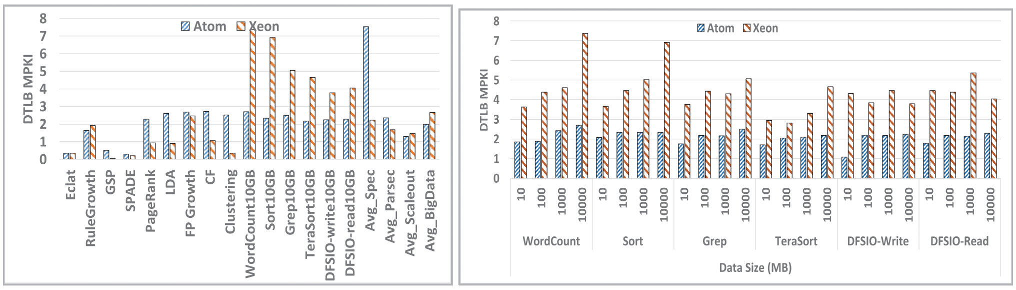


Fig. 23. DTLB MPKI. (a) Big data workloads. (b) Different datasets of Hadoop micro-benchmarks.

big data applications have the highest ITLB and DTLB MPKI compared to traditional benchmarks. In contrast, SPEC is experiencing the highest ITLB and DTLB MPKI on Atom. Moreover, Atom has large ITLB MPKI as compared to Xeon, but lower DTLB MPKI in comparison to Xeon. Figures 22(b) and 23(b) show 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 solution. Similar results were obtained in a closely related work (Wang et al. 2014) on Xeon. Also, the results show that the size of data affects DTLB miss rates in both architectures.

6 DISCUSSION

In this section, based on the results and discussions throughout the article, the key findings are presented as follows:

6.1 Application Analysis

- Compute-bound applications remains almost the same on Xeon and Atom.
- Due to low processing capacity of little Atom, applications that are classified as I/O bound on Xeon illustrate hybrid behavior on Atom.
- CPU user utilization difference between Atom and Xeon is small for reduce phase of the Hadoop applications. Thus, reduce phase clearly favors the little Atom server, considering the effective core utilization and lower power consumption of Atom.

6.2 System Analysis

- Traditional CPU applications are optimized for big Xeon core. As IPC results show significantly larger performance drop of 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 server is more energy-efficient compared to little core server for traditional applications.
- Little-core-based server is more efficient in terms of energy efficiency for big data processing with the exception of the I/O application.
- The increase in the size of data and performance constraints makes big-core server an efficient choice. For the smaller data size, the higher processing capability of big core hides the

I/O communication cost effectively. However, the rise in data size aggravates the I/O cost to an extent that it diminishes the benefit of the high processing capability of the big core for I/O intensive benchmarks.

6.3 Post-acceleration CPU Code Analysis

- Unlike our initial prediction of processing the post-accelerated big data workload on big core, choice of big vs. little before and after accelerations is different for applications. We mostly found that Hadoop-based big data applications prefer little-core post-acceleration, while other applications like SPMF mostly prefer big-core servers.

6.4 Microarchitecture Analysis

- Big data application has higher instruction cache MPKI than the traditional applications. Instruction footprints of the big data applications do not fit in the L1 instruction cache on both Xeon and Atom.
- Big data applications have high branch misprediction rate compared to traditional applications. Traditional CPU and parallel benchmarks have instruction working sets with tight loops, therefore making them easy for the branch predictor to predict correctly and reduce their misprediction rate.
- 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 core servers. Thus, big data applications require efficient instruction prefetchers to predict complex patterns and more accurate branch predictor to handle the unknown control flow.
- The size of data has a non-trivial impact on several micro-architecture parameters on both Atom and Xeon servers.
- A small two-level data cache is sufficient for big data processing on little core; however, the instruction cache hierarchy pipeline design needs improvement.
- On Xeon, big data applications have an order of magnitude lower ITLB miss rate compared to Atom. Little core needs architectural improvement in instruction TLB miss overhead management.

7 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 platforms. 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 (Arora et al. 2012). The most prominent big data benchmarks are HiBench, CloudSuite, BigDataBench, CloudCmp, LinkBench, TPC, BigBench, and CloudRank-D. HiBench (Huang et al. 2010) is a benchmark suite for Hadoop MapReduce developed by Intel. The CloudSuite (Ferdman et al. 2012, 4) benchmark was developed for Scale-Out cloud workloads focusing on web services and off-line analytics. CloudSuite benchmark demonstrates that current existing computer systems are not satisfying the scale-out workloads requirements. BigDataBench (Wang et al. 2014; Gao et al. 2013) was developed very recently and includes online service and offline analytics for web service applications to characterize big data applications. This work is a closely related to ours; however, they mainly performed characterization on high-end Xeon servers. Our work is different as it attempts to compare the two server architectures for big data processing in terms of energy-efficiency, performance, and data processing capacity. CloudCmp (Li et al. 2010) uses 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 (Armstrong et al. 2013).

The Transaction Procession Performance Council (TPC) has released several benchmark suites in recent years, including TPC-C, TPC-E, and TPC-DS for online transaction processing. BigBench (Ghazal et al. 2013) is a new big data benchmark that adopts TPC-DS as its basis and expands it for offline analytics on Xeon high-performance server. CloudRank-D (Luo et al. 2012) benchmark is developed for cloud computing that focuses on the capacity planning and system evaluation by modeling the cloud computing usage configurations. In addition, authors in Jia et al. (2017) have investigated the top-Down analysis method to characterize their micro-architectural characteristics of the big data analytics workloads using top-down analysis on the Intel Xeon E5-2680 v3 server. Jia et al. (2014) summarize the impact of software stacks of big data application on Intel Xeon E5645. Although this work is related to ours, they mainly performed characterization on high-end Xeon servers. Our work is different as it attempts to compare the two server architectures for big data processing in terms of energy-efficiency, performance, and data processing capacity as well as micro-architecture analysis.

Several prior researches have characterized traditional CPU and parallel applications such as SPEC2006, PARSEC, and NAS on high-performance server-class processors (Prakash et al. 2008). However, big data applications illustrated fundamentally different characteristics from the traditional CPU and parallel applications. It is therefore vital to 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 comparison with big data workloads.

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

There have been also several research studies on application-specific (Yu et al. 2007; Yu et al. 2004) and domain-specific accelerators (Arnold et al. 2001; Li et al. 2009; Arora et al. 2010). Using tightly integrated FPGA (Luo et al. 2013) with CPU, and GPU with CPU (Baru et al. 2012), to accelerate big data processing, have been proposed in recent work. FPGAs have also recently been used to accelerate intelligent personal assistant (IPA) workloads in datacenters. Johann et al. (2015) have used a specific intelligent personal assistant application, Sirius, where FPGA is explored for the implementation of core functions of Sirius in data centers and reducing the query latency. 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. In our article, we did not indulge on how to implement and accelerate the offload part of the application on FPGA; we have evaluated how offloading the hotspot map tasks to FPGA affects the choice of big- vs. little-core-based servers to process the code that remains on the CPU. To the best of our knowledge, the only work on this topic is by Arora et al. (2012), 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 it benefits less from multiple cores, as there is less TLP after GPU offloading. In this article, we demonstrated how deploying an accelerator for big data affects the choice of big vs. little core for efficient processing.

In addition, there are several recent researches that investigated the data access pattern, system resource utilization and HDFS bandwidth of big data applications. Hibench (Huang et al. 2010) demonstrates the resource utilization and HDFS bandwidth of various Hadoop applications.

Phi-Sched (Krish et al. 2014) studies the behavior of Hadoop applications and proposes a hardware-aware workflow scheduler for Hadoop by analyzing the CPU, memory, and storage and network usage of Hadoop applications. Our work is different from the above works as it not only includes Hadoop representative applications, but it also studies non-Hadoop-based big data applications. Additionally, we characterize these benchmarks from system-level perspective to highlight the choice of server architecture for energy-efficient processing of big data. It also performs a comprehensive micro-architectural level analysis, which helps provide insights where big data applications require improvements in the current server architecture design.

8 CONCLUSION

In this article, we presented a comprehensive application, system, and microarchitecture-level analysis of big data applications on two distinct server platforms: the conventional server design, which uses a high-performance big Xeon core, and the new trajectory in server design, which uses a low-power little Atom core to address the dark silicon challenge.

We evaluated the resource utilization of big data applications to understand their runtime behavior. The recourse utilization analysis results show that a big data application can be classified differently in terms of being compute-intensive, I/O-intensive, or hybrid on big Xeon core and little Atom core. They also show significantly larger performance drop of 37%, on average, for big data applications compared to traditional CPU applications when running on big-core server compared to little core server, indicating while big Xeon core has been optimized for traditional CPU applications, it is not optimized for big data applications. While big-core server is more energy-efficient compared to little-core server for traditional applications, for big data, it is less efficient. Little-core-based server is more efficient in terms of EDP for big data processing with small data sizes compared to big-core-based server. However, as the size of data increases and with performance constraints, big-core server becomes an efficient choice. The analysis of data processing capability and efficiency of big data applications illustrates that the choice of big-core-based 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 the most efficient server architecture to process the remaining code after acceleration. The results show that there is a clear difference between the choices of big- vs. little-core-based servers before and after acceleration. While most benchmarks clearly favor little core post-acceleration, several benchmarks show higher speed-up on big-core over little-core post-acceleration compared to pre-acceleration.

To provide insights on whether current server design based on big and little core architectures requires improvement in their microarchitecture parameters for efficient big data processing, we performed a comprehensive microarchitecture characterization and compared the results with traditional Spec, PARSEC, and scale-out applications. Our analysis indicates that the size of data has a non-trivial impact on several micro-architecture parameters on both Atom and Xeon servers corroborating recent observations on Xeon. Moreover, results show that while a small 4x1MB two-level data cache is sufficient for big data processing on little core, the instruction cache hierarchy pipeline design 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-core servers. Moreover, big data applications require efficient instruction prefetchers to predict complex patterns and more accurate branch predictor to handle the unknown control flow.

ACKNOWLEDGMENT

We are grateful for the stimulating discussions and comments by Dr. Avesta Sasan.

REFERENCES

Accelerating Hadoop* applications using Intel QuickAssist tech. 2013.

D. G. Andersen et al. 2009. FAWN: A fast array of wimpy nodes. In *the Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles (SOSP'09)*.

R. Appuswamy et al. 2013. Scale-up vs. scale-out for Hadoop: Time to rethink? In *Proceedings of the 4th annual Symposium on Cloud Computing*. ACM, 20.

T. G. Armstrong, Vamsi Ponnekanti, Dhruba Borthakur, and Mark Callaghan. 2013. LinkBench: A database benchmark based on the Facebook social graph. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*. ACM, 1185–1196.

Nidhi Arora, Kiran Chandramohan, Nagaraju Pothineni, and Anshul Kumar. 2010. Instruction selection in asip synthesis using functional matching. In *23rd International Conference on VLSI Design, 2010 (VLSID'10)*. IEEE, 146–151.

Arora Manish et al. 2012. Redefining the role of the CPU in the era of CPU-GPU integration. *IEEE Micro*. 32, 6 (2012), 4–16.

M. Arnold et al. 2001. Designing domain-specific processors. In *Proceedings of the 9th International Conference on Codesign and System Synthesis (CODES'01)*. ACM.

L. A. Barroso et al. 2013. The datacenter as a computer: An introduction to the design of warehouse-scale machines. *Synth. Lect. Comput. Architect.* 8, 3 (2013), 1–154.

Chaitanya Baru, Milind Bhandarkar, Raghunath Nambiar, Meikel Poess, and Tilmann Rabl. 2012. Setting the direction for big data benchmark standards. In *Technology Conference on Performance Evaluation and Benchmarking*. Springer, Berlin, Heidelberg, 197–208.

E. Blem et al. 2013. Power struggles: Revisiting the RISC vs. CISC debate on contemporary ARM and x86 architectures. In *Proceedings of the Conference on High-Performance Computer Architecture (HPCA'13)*.

Collaborative Filtering. Retrieved 2017 from http://archive.cloudera.com/cdh5/cdh/5/mahout/mahout-core/org/apache/mahout/cf/taste/hadoop/item/package-tree.html.

M. Dimitrov et al. 2013. Memory system characterization of big data workloads. *Proceedings of the IEEE International Conference on Big Data*.

Dstat. Retrieved 2018 from http://lintut.com/dstat-linux-monitoring-tools/.

M. Ferdman et al. 2012. Clearing the clouds: A study of emerging scale-out workloads on modern hardware. In *Proceedings of the ACM SIGPLAN Conference*.

Frequent Itemset Mining Dataset Repository. Retrieved 2017 from http://fimi.ua.ac.be/data/.

W. Gao et al. 2013. Bigdatabench: A big data benchmark suite from web search engines. In *Proceedings of the Conference on Architectures and Systems for Big Data (ASBD'13) in Conjunction with the International Symposium on Computer Architecture (ISCA'13)*.

A. Ghazal et al. 2013. Bigbench: Towards an industry standard benchmark for big data analytics. In *Proceedings of the ACM SIGMOD Conference*.

A. Gutierrez et al. 2014. Integrated 3D-stacked server designs for increasing physical density of key-value stores. In *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'14)*.

Nikos Hardavellas, Michael Ferdman, Babak Falsafi, and Anastasia Ailamaki. 2011. Toward dark silicon in servers. *IEEE Micro* 31, 4 (2011), 6–15.

H. Homayoun et al. 2012. Dynamically heterogeneous cores through 3D resource pooling. In *Proceedings of the Conference on High-Performance Computer Architecture (HPCA'12)*.

S. Huang et al. 2010. The hibench benchmark suite: Characterization of the mapreduce-based data analysis. In *Proceedings of the 26th International Conference on Data Engineering Workshops (ICDEW'10)*.

K. Hwang et al. 2016. Cloud performance modeling with benchmark evaluation of elastic scaling strategies. *IEEE Trans. Parall. Distrib. Syst.* 27, 1 (2016), 130–143.

Intel VTune Amplifier XE Performance Profiler. 2015. Retrieved from http://software.intel.com/en-us/articles/intel-vtune-amplifier-xe/.

Z. Jia et al. 2014. Characterizing and subsetting big data workloads. In *Proceedings of the IEEE International Symposium on Workload Characterization (IISWC'14)*.

Z. Jia et al. 2017. Understanding big data analytics workloads on modern processors. *IEEE Trans. Parall. Distrib. Syst.* 28, 6 (2017), 1797–1810.

H. Johann et al. 2015. Sirius: An open end-to-end voice and vision personal assistant and its implications for future warehouse scale computers. *ACM SIGPLAN Notices* 50, 4, 223–238. ACM, 2015.

R. T. Kaushik et al. 2010. Greenhdfs: Towards an energy-conserving, storage-efficient, hybrid Hadoop compute cluster. *Proceedings of the USENIX Annual Technical Conference.*

T. Khavari et al. 2014. Energy-efficient mapping of biomedical applications on domain-specific accelerator under process variation. In *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED'14).*

V. Kontorinis et al. 2012. Managing distributed UPS energy for effective power capping in data centers. In *Proceedings of the 39th International Symposium on Computer Architecture (ISCA'12).*

V. Kontorinis et al. 2014. Enabling dynamic heterogeneity through core-on-core stacking. In *Proceedings of the the 51st Annual Design Automation Conference (DAC'14).*

K. R. Krish, Ali Anwar, and Ali R. Butt. 2014. [phi] Sched: A heterogeneity aware Hadoop workflow scheduler. In *Proceedings of the International Symposium on Modeling Analysis and Simulation of Telecommunication Systems (MASCOTS'14).*

James T. Kukunas, V. Gopal, J. Guilford, S. Gulley, A. van de Ven, and W. Feghali. 2014. High performance ZLIB compression on Intel® architecture processors. *White paper.*

A. Li et al. 2010. CloudCmp: Comparing public cloud providers. *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement.*

Li Haoyuan et al. 2008. PFP: Parallel FP-growth for query recommendation. *Proceedings of the ACM Conference on RecOmmender Systems.*

T. Li et al. 2009. Fast enumeration of maximal valid subgraphs for custom instruction identification. *Proceedings of the Conference on Automation Science and Engineering (CASES'09).*

F. Liang et al. 2014. Performance characterization of Hadoop and data MPI based on Amdahl's second law. In *Proceedings of the 9th IEEE International Conference on Networking, Architecture, and Storage (NAS'14).* 207–215.

K. Lim et al. 2008. Understanding and designing new server architectures for emerging warehouse-computing environments. *ACM SIGARCH Comput. Architect. News* 36, 3, 315–326.

Z. Lin and P. Chow. 2013. Zcluster: A zynq-based Hadoop cluster. In *Proceedings of the International Conference on Field Programmable Technology (FPT'13).* 450–453.

Luo Chunjie et al. 2012. Cloudrank-d: Benchmarking and ranking cloud computing systems for data processing applications. *Front. Comput. Sci.* 6, 4 (2012), 347–362.

Mahout: Scalable machine-learning and data-mining library. Retrieved 2017 from http://mahout.apache.org/.

M. Malik and H. Homayoun. 2015. Big data on low power cores: Are low power embedded processors a good fit for the big data workloads? In *Proceedings of the 33rd IEEE International Conference on Computer Design (ICCD'15).* 379–382.

M. Malik et al. 2015a. System and architecture level characterization of big data applications on big and little core server architectures. In *Proceedings of the IEEE International Conference on Big Data.* 85–94.

M. Malik et al. 2015b. Characterizing Hadoop applications on microservers for performance and energy efficiency optimizations. In *Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS'16).* 153–154.

S. Nilakantan et al. 2013. Platform-independent analysis of function-level communication in workloads. In *Proceedings of the IEEE International Symposium on Workload Characterization (IISWC'13).*

K. Ousterhout et al. 2015. Making sense of performance in data analytics frameworks. In *Proceedings of the Symposium on Networked Systems Design and Implementation (NSDI'15).* 293–307.

V. J. Reddi et al. 2010. Web search using mobile cores: Quantifying and mitigating the price of efficiency. In *Proceedings of the ACM SIGPLAN Conference.*

H. Sayadi, D. Pathak, I. Savidis, and H. Homayoun. 2018. Power conversion efficiency-aware mapping of multithreaded applications on heterogeneous architectures: A comprehensive parameter tuning. In *Proceedings of the 23rd Asia and South Pacific Design Automation Conference (ASP-DAC'18).* IEEE, 70–75.

Y. Shan et al. 2010. FPMR: Mapreduce framework on FPGA. In *Proceedings of the Annual ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA'10).*

SPMF. Retrieved 2016 from http://www.philippe-fournier-viger.com/spmf/index.php.

T. Honjo and K. Oikawa. 2013. Hardware acceleration of Hadoop mapreduce. In *Proceedings of the IEEE International Conference on Big Data.* 118–124.

T. K. Prakash et al. 2008. Performance characterization of SPEC CPU2006 Benchmarks on Intel Core 2 Duo Processor. In *Proceedings of the International Seminar on Aerospace Science and Technology (ISAST'08).*

J. Veiga et al. 2016. Performance evaluation of big data frameworks for large-scale data analytics. In *Proceedings of the IEEE International Conference on Big Data.*

L. Wang et al. 2014. Bigdatabench: A big data benchmark suite from internet services. In *Proceedings of the 20th International Symposium on High Performance Computer Architecture (HPCA'14).* IEEE.

WattsUpPro power meter. 2015. Retrieved from https://www.wattsupmeters.com/secure/index.

T. L. Willke et al. 2012. Graphbuilder—A scalable graph construction library for Apache Hadoop. In *Big Learning Workshop on Neural Information Processing Systems (NIPS'12).*

Xi Luo, Walid Najjar, and Vagelis Hristidis. 2013. Efficient near-duplicate document detection using FPGAs. In *2013 IEEE International Conference on Big Data*. IEEE, 54–61.

P. Yu et al. 2007. Disjoint pattern enumeration for custom instructions identification. In *Proceedings of the Conference on Field Programmable Logic (FPL'07)*. 273–278.

P. Yu et al. 2004. Scalable custom instructions identification for instruction-set extensible processors. In *Proceedings of the Conference on Automation Science and Engineering (CASES'04)*.

Received November 2017; revised March 2018; accepted May 2018