doi:10.3772/j.issn.1006-6748.2017.03.003

Characterizing big data analytics workloads on POWER8 SMT processors[①]

Jia Zhen (贾　禛)[*][**], Zhan Jianfeng[②][*][**], Wang Lei[*], Zhang Lixin[*]
([*] State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100080, P. R. China)
([**] University of Chinese Academy of Sciences, Beijing 100049, P. R. China)

Abstract

Big data analytics is emerging as one kind of the most important workloads in modern data centers. Hence, it is of great interest to identify the method of achieving the best performance for big data analytics workloads running on state-of-the-art SMT (simultaneous multithreading) processors, which needs comprehensive understanding to workload characteristics. This paper chooses the Spark workloads as the representative big data analytics workloads and performs comprehensive measurements on the POWER8 platform, which supports a wide range of multithreading. The research finds that the thread assignment policy and cache contention have significant impacts on application performance. In order to identify the potential optimization method from the experiment results, this study performs micro-architecture level characterizations by means of hardware performance counters and gives implications accordingly.

Key words: simultaneous multithreading (SMT), workloads characterization, POWER8, big data analytics

0 Introduction

Simultaneous multithreading (SMT) has been introduced to modern processor designs for over a decade[1]. It permits instructions from more than one thread to be executed in any given pipeline stage at a time and delivers better processor resource utilizations by executing instructions of other threads when there is a pipeline stall. For the SMT technology has the ability to improve the parallelization of computations and shows the potential to increase system throughput, it has been adopted in most of current generations of microprocessors. The state-of-the-art IBM POWER8 processor even doubles the hardware thread parallelism in comparison with its predecessor (i. e., POWER7 +) and supports up to 8-way multithreading (SMT8)[2].

Many studies have proved that SMT can substantially increase the performance of various types of applications[3-5]. However how big data analytics applications, which are emerging as one of the major workloads in modern data centers, performing on SMT processors has not been well investigated. Since Spark[6] is gaining a wide industry adoption in big data domain due to its superior performance, simple interfaces, and a rich library for analysis and calculation, this work chooses the Spark based workloads as the representative big data analytics workloads and presents measurements on the POWER8 platform. By tuning the SMT modes among ST (single-threaded), SMT2, SMT4 and SMT8, this study investigates the processor behaviors using hardware performance counters. Experimental results show that the thread assignment policy and cache contention play important roles in application performance.

This study is a part of a large project of dynamically auto-tuning SMT modes to accelerate big data analytics workloads in modern data centers. The observations from workload characterizations can help design of such a project. The observations and findings in this paper are summarized as follows:

1) The thread assignment policy has a significant performance impact. An up to 9.6 × performance gap exists between the proper one and the improper one.

2) The SMT technology can achieve performance gains (up to 33%) for big data analytics workloads and different workloads favour diverse SMT modes.

3) Cache contentions may obviate the performance improvements brought by the SMT technology.

The remainder of the paper is organized as fol-

① Supported by the National High Technology Research and Development Program of China (No. 2015AA015308) and the State Key Development Program for Basic Research of China (No. 2014CB340402).

② To whom correspondence should be addressed. E-mail: zhanjianfeng@ict.ac.cn

Received on Mar. 4, 2016

lows. Section 1 states the hardware and software backgrounds. Section 2 lists the experimental methodology. Section 3 presents the micro-architectural characteristics of the Spark workloads. Section 4 discusses the related work and Section 5 concludes the paper.

1 Background

1.1 POWER8 processor

POWER8[2] provides many new features compared with its predecessor (i. e., POWER7 +). It doubles the hardware thread parallelism and supports up to 8-way multithreading. The eight hardware threads in a POWER8 processor are split into two thread-sets: the even thread-set (T0, T2, T4, and T6) and the odd thread-set (T1, T3, T5, and T7). Each thread-set occupies half of the issue queue entries and half of the execution pipelines. The POWER8 processor improves thread-switch time and removes thread placement restrictions. That is to say, any software thread can be run on any thread-set.

The POWER8 architecture, like most modern processors, can dispatch groups of machine instructions every cycle. Groups are dispatched into the execution pipeline in order and completed in order. Instructions or internal operations in a group are executed out of order, and all instructions in a group must be finished before the group retired.

1.2 Spark

Spark[6] is a general in memory computing engine for big data processing. The basic idea of Spark is keeping the working set of data in the memory. The proposed abstraction resilient distributed dataset (RDD) provides a restricted form of shared memory, based on coarse grained transformations rather than fine-grained updates to share state so as to achieve fault tolerance efficiently.

The Spark cluster works in a master-slave mode. The Spark application needs to connect to the cluster master, which allocates resources across applications. Once connected, Spark acquires workers in the cluster, which are processes that run computations and store data for the Spark application. The workers are long-lived processes that can store RDD partitions in RAM across operations[6]. By default, the Spark framework allocates one worker instance per server in the cluster and assigns all cores to the worker instance.

The Spark shows partitioned parallelism so as to process big data efficiently. By partitioning the input data set, each Spark job is split into many independent tasks, working on a part of the data in parallel. Each task can be given to a hardware thread to execute in parallel with the other hardware threads and there is no data racing among them. So the Spark system has the potential to scale with the number of hardware threads.

2 Experimental methodology

2.1 System Configuration

IBM POWER S822L (8247-22L) server is used in this paper. The S822L server is based on POWER8 technology and it includes four sockets. Each socket has six out-of-order cores with speculative pipelines. Table 1 lists the main hardware configurations of the server. The server runs Ubuntu 15.04 with the Linux kernel version 3.19.0. One of the cores has been deconfigured due to hardware failures. So in the system, there are 23 cores online and up to 184 hardware threads (in SMT8 mode).

Table 1 Hardware configurations

CPU type	IBM POWER8
# Cores	24
# Threads per core	8
#Sockets	4
Architecture	ppc64le
Byte Order	Little Endian
L1 Data Cache	64kB
L1 Instruction Cache	32kB
L2 Cache (unified)	512kB
L3 Cache (unified)	8 MB per core
Memory	256 GB
Disk	4 × 1 TB

2.2 Benchmarks

Five Spark based workloads from BigDataBench[7] are characterized in this section. Table 2 presents the problem size and the corresponding data type for each of them. This paper deploys a Spark cluster in standalone mode with Spark version 1.0.2 and uses HDFS provided by Hadoop version 1.2.1. IBM J9 is used as the Java runtime environment implementation, which supports Java version 1.7.0.

Table 2 Big data analytics workloads

Workload	Problem size	Data type
Sort	17.5GB	sequence file
Wordcount	17.5GB	text
Grep	17.5GB	text
Naive Bayes	15.7GB	text
PageRank	16.5GB	graph

2.3 Performance data

Wall-clock time is used to determine the execution time. And hardware performance counters are used to understand the processor pipeline behaviors. Perf[8], a profiling tool for Linux 2.6+ based system, is used to access the hardware performance counters by specifying the performance event numbers and corresponding unit masks. This paper collects about 10 events which can be found in Ref. [9].

3 Measurements and findings

The SMT technology has double-sided affects. It can hide short duration pipeline stalls, but may increase the pressure of resource allocations in pipelines. For instance, when instructions from one thread are delayed waiting for data because of cache misses, instructions from other threads can continue to execute. However, it may increase the overall cache misses for cache contentions. In this paper, both sides are observed. At the same time, the study finds that the thread assignment policy will affect performance significantly. The following parts of this section show the performance data of diverse thread assignment policies and SMT modes.

3.1 Single worker instance

The experiments are first running with the default Spark thread assignment policy: allocating one worker instance (i. e., JVM) for each server and assigning all cores to the worker instance. So when the SMT mode is changed, the number of hardware thread that can be used by the system is also changed.

1) Speedup

Fig. 1 shows the performance of Spark applications among different SMT modes. The data processing speed

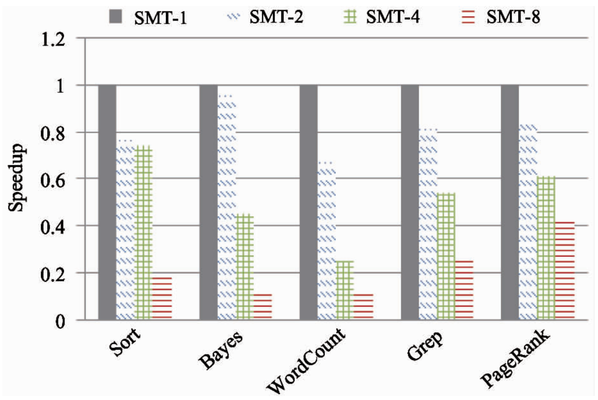

Fig. 1 Single worker instance SMT speedup

of the ST mode (i. e., single threaded) is used as the baseline. The figure shows the normalized processing speed as the performance (speed up). Contrasted with expected results, the SMT technology does not achieve any performance gain. On the contrary, the SMT technology seems to degrade the performance: the higher SMT modes the more performance degradations.

2) CPI Stack

In order to find what happens in the pipeline when different SMT modes are used, the processor pipeline statistics are collected by accessing hardware performance counters.

Fig. 2 illustrates the normalized CPI stacks of POWER8 cores on different SMT modes. The CPI stack apportions the total CPU time (cycles) among the five parts in the execution pipeline on a per-thread basis as shown in Fig. 2. An instruction group can contain up to eight PPC (Power PC) instructions. When they are all finished, the group entry in the global completion table (GCT) is removed and the next group is eligible for computation. These cycles are referred to as PM_GRP_CMPL. If there is no new instruction dispatched for this thread, the global completion table will have no slot for the thread. These cycles are referred to as PM_GCT_NOSLOT_CYC, which are also known as front end stalls. The computation stall cycles (PM_CMPLU_STALL) are caused by the limited computation resources or less of data. The PM_CMPLU_STALL_THRD also means computation stall, however, it is due to thread conflicts. That is to say that an instruction group is ready to be executed but it is another thread's turn. The cycles after all instructions have finished to group completed are referred to as PM_NTCG_ALL_FIN.

Within the five main parts of CPI stack in Fig. 2, only the PM_GRP_CMPL indicates instruction groups completed, and all others mean the pipeline stalled due to certain reasons. The PM_GRP_CMPL has less proportion when higher SMT mode is used, which indicates that more cycles are needed for each retired instruction (i. e., higher thread CPI). The component that owns the most proportion is PM_CMPLU_STAL (computation stall). That is to say that the computation stall is the main bottleneck. The proportion of PM_CMPLU_STALL_THRD does not have large fluctuations among different SMT modes for Spark workloads. And the proportion of PM_CMPLU_STALL_THRD does not increase with the SMT mode, which indicates that higher SMT mode's performance degradation is not caused by threads conflicts.

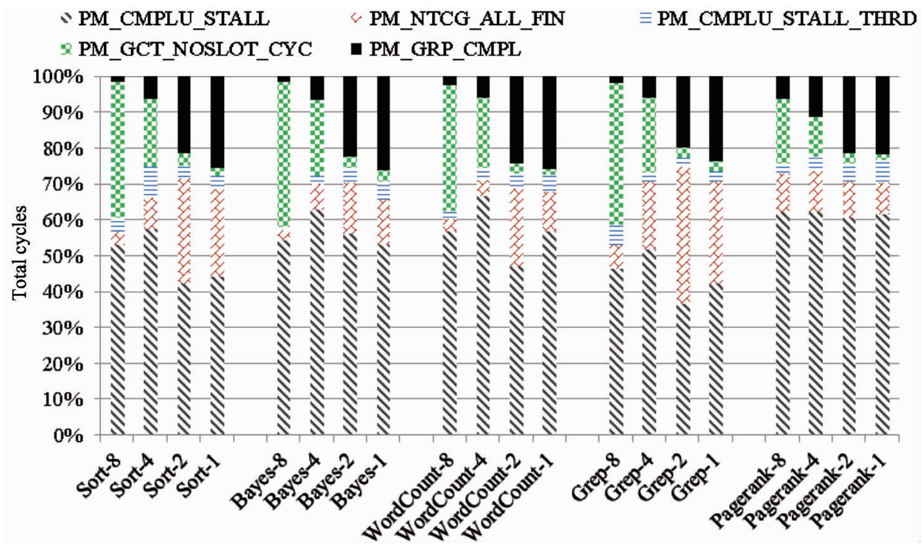

Fig. 2 CPI stacks of single worker instance

The component that increases with SMT mode is the PM _ GCT _ NOSLOT _ CYC, which indicates pipeline front end stall. It happens before the instruction entering the out-of-order computation units, which may be caused by instruction cache misses, branch miss predictions, etc. In the figure, a high SMT mode increases the pipeline front end pressures of the POWER8 processor and the front end stall is the main reason that causes performance delegations from the perspective of micro-architecture. other reasons may also contribute to this phenomenon, which are out of the scope of this work.

3.2 Multiple worker instances

Section 3.1 uses the default Spark thread assignment policy, which forks 184 threads in one single JVM (Java virtual machine) in SMT8 mode. Considering the fact that multiple moderate JVMs may beat a single huge JVM as discussed in Apache Spark user list[10], another thread assignment policy is adopted to configure the Spark environment. This subsection generates 23 worker instances (i. e., 23 JVMs) and assigns each physical core to each worker instance. The number of hardware threads is changed for each worker instance according to the SMT mode.

1) Speedup

In order to investigate the impact of thread assignment policy, the performance between the single worker instance and 23 worker instances in SMT8 mode is compared. Fig. 3 shows the data processing speed normalized to single worker instance in SMT8 mode for each workload. There are more than 2 times performance gaps between single worker instance and multiple worker instances. For Bayes, the performance gap is about 9.6 times.

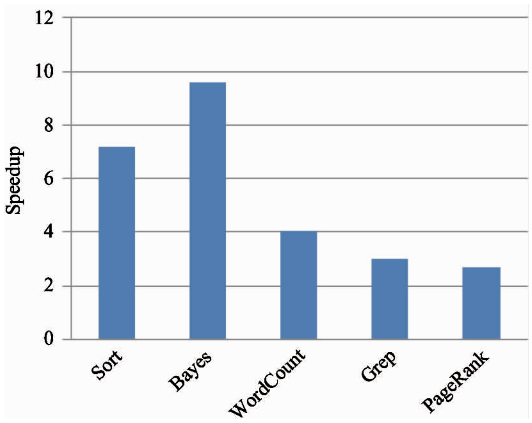

Fig. 3 Performance gaps between single worker instance and multiple worker instances in SMT8 mode

Fig. 4 illustrates the performance of different SMT modes by using the ST mode processing speed as the baseline when using multiple worker instances. Totally different phenomena from the single worker instance situation in Section 3.1 have been found. The SMT technology can achieve up to 33% performance gains in comparison with ST mode and different workloads favor diverse SMT modes. For all the Spark based workloads investigated in this paper, the SMT8 mode never achieves the best performance, which may be caused by a number of reasons, e. g., instruction prefetching is disabled for SMT8 so as to reduce cache misses and the inner core may not have enough resources to maintain the entire architecture state of the 8 threads in each core[2].

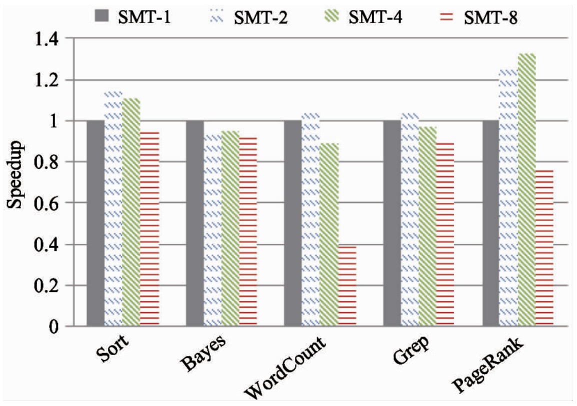

Fig. 4 Multiple worker instances SMT Speedup

There is a huge performance gap between the best SMT mode and the worst one, which makes the SMT tuning attractive. In addition, tuning the SMT mode should also take the thread assignment policy into consideration. A single huge JVM, which is assigned with lots of threads (184 threads in this paper), may incur overheads and cause performance degradations (up to 8.6 times in this paper). For the system owns lots of hardware threads, assigning multiple JVMs with a small number of threads in each of them is suggested. However, more JVMs will take up more resources and have starting overheads. So there is a trade off between the number of processes (JVMs) and how many threads in each process. For our experiments, the 23 JVMs (i. e., each physical core owning a JVM) can achieve good performance, but for other systems, whether the rule can achieve a pretty good performance is still needed to investigate in the further work.

2) CPI stack

Fig. 5 shows the CPI stacks for each workload running in different SMT modes when using multiple worker instances. In the figure, higher SMT mode increases resource utilization and incurs more pipeline stalls, especially for SMT8 and SMT4 modes. Even there are more pipeline stalls, the overall performance can also be improved with higher SMT modes (Fig. 4) because more hardware threads are activated. And multiple worker instances really decrease the pipeline front end pressures, which is represented by less PM _ GCT _ NOSLOT _ CYC proportions in comparison with the single worker instance situation (Fig. 2). Even though the proportion of PM _ GCT _ NOSLOT _ CYC is still rising with the increasing of thread number in each core for some workloads, the trend is much moderate in comparison with the single worker instance situation.

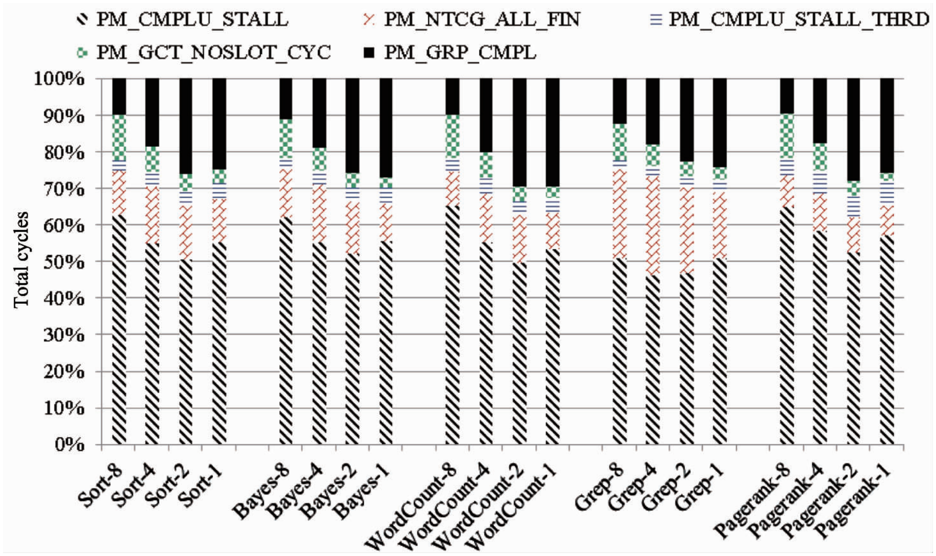

Fig. 5 CPI stacks of multiple worker instances

Multiple threads in a single JVM increases the pressure of pipeline front end and obviates performance improvements brought by the SMT technology. Those phenomena indicate that the pipeline front end stall (GCT no slot stall) may be used to guide the tuning of thread assignment policy. A high percentage of front end stall indicates that more moderate JVMs are welcomed.

3) Cache behaviors

In this subsection, cache behaviors are investigated for cache performance is heavily affected by the SMT technology. Fig. 6 and Fig. 7 show the L1 data cache misses per kilo instructions (MPKI) due to load instructions and store instructions respectively. The SMT technology really aggravates cache contentions. The higher SMT modes the more cache misses. The only exceptions are the Sort and Grep's store misses. A high SMT mode can decrease the cache misses, which might be caused by data sharing or effective prefetching. This is a phenomenon of constructive cache interferences as mentioned in Ref. [3]. For the SMT technology can multiplex the cache miss cycles by issuing instructions from other threads to the pipeline, a slight increase of cache misses will not affect the performance improvements. Taking Sort as an example, the workload running in SMT2 mode owns more cache misses than in ST mode, but the proportion of computation stall (PM _ CMPLU _ STALL) is less than the ST counterpart in Fig. 5. The Sort achieves better performance in SMT2 mode than in ST mode. Similar phenomena can be observed from other workloads.

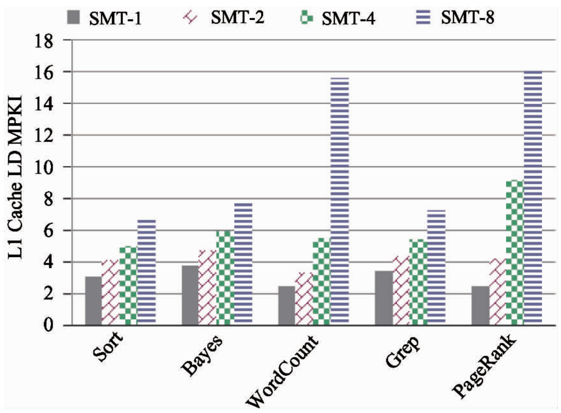

Fig. 6 L1 data cache misses per kilo instructions caused by load instructions

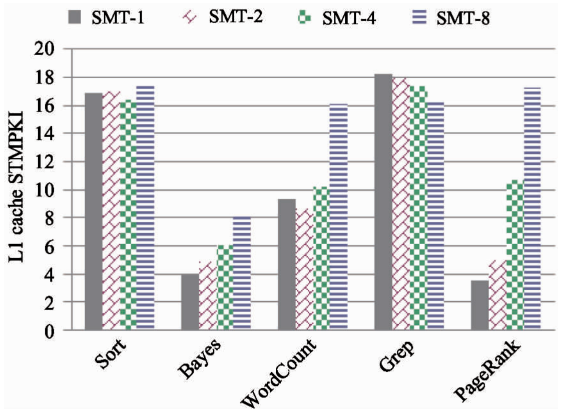

Fig. 7 L1 data cache misses per kilo instructions caused by store instructions

However, a large amount of increased cache misses, which indicates severe cache conflicts, will decrease performance. This is caused by too many threads competing for cache. The data fetched by one thread can be flushed by the other thread and the inherent locality is destroyed in some degree. So the delay that the execution engine gets the missed data is not hidden by more threads but prolonged. WordCount and PageRank seem to confront this situation. They get severe performance degradations when switching from SMT4 to SMT8 in Fig. 4. In Fig. 6 and Fig. 7, when running in SMT8 mode, they have sharp increases in data cache misses, including both loads and stores, in comparison with the SMT4 mode. The L2 cache and L3 cache statistics are also inspected and similar tendencies are found.

The SMT technology may incur cache contentions. A slight increase of cache misses will not affect the performance. However severe cache conflicts will result in performance degradations. So cache contentions should be taken into consideration as a key factor when tuning the SMT mode.

4 Related work

Lots of work characterizes big data workloads running on modern super-scalar processors by using hardware performance counters[7,11,12]. However none of them presents the performance impact of SMT technology. There are also many studies that characterize workload behaviors on SMT platforms[3,4,5,13,14]. Huang, et al.[3], Bulpin, et al.[13] and Mathis, et al.[4] examine the characterizations of traditional applications running on Intel Hyper-Threading supported processors and IBM SMT supported POWER5 processors. Ruan et al.[5] perform macroscopic analysis as well as micro-architectural measurements to understand the origins of the performance bottlenecks for SMT processors on network servers. Zhang, et al.[14] present a methodology that enables precise performance interference predictions on SMT processors. However none of them focuses on big data analytics workloads with platforms adopting SMT8. For the microprocessor architecture has been evolving, workload characteristics are also changing. To our best knowledge, no previous work has investigated the SMT technology impacts of big data analytics workloads, especially on SMT8 platforms.

5 Conclusion and further work

In this paper, by means of hardware performance counters, five representative Spark workloads running on the state-of-the-art POWER8 platform are characterized and the platform owns a wide range of tuneable SMT modes. This paper finds that:

1) The thread assignment policy has a significant impact on application performance and an improper thread assignment policy increases the pressure of pipeline front end.

2) The SMT technology can achieve performance gains for Spark workloads and different workloads favour diverse SMT modes.

3) Cache contentions may obviate the performance improvements brought by the SMT technology.

This study is a part of a large project of auto-tuning SMT modes to accelerate big data analytics systems. Of relevance to this paper is the development of a tuning tool that can adjust SMT modes dynamically. The knowledge from this paper can guide us to develop such a tool. We are using data from hardware performance counters to predict the suitable SMT modes for big data analytics workloads automatically. It is believed that a dynamic, automatic SMT mode tuning tool has attractive potentials to boost the performance of big data analytics workloads.

References

[1] Tullsen D M, Eggers S J, Emer J S, et al. Exploiting choice: instruction fetch and issue on an implementable simultaneous multithreading processor. *ACM SIGARCH Computer Architecture News*, 1996, 24(2): 191-202

[2] Sinharoy B, Van Norstrand J A, Eickemeyer R J, et al. IBM POWER8 processor core microarchitecture. *IBM Journal of Research and Development*, 2015, 59(1): 1-21

[3] Huang W, Lin J, Zhang Z, et al. Performance characterization of java applications on SMT processors. In: Proceedings of 2005 IEEE International Symposium on Performance Analysis of Systems and Software, Austin, USA, 2005. 102-111

[4] Mathis H M, Mericas A E, McCalpin J D, et al. Characterization of simultaneous multithreading (SMT) efficiency in POWER5. *IBM Journal of Research and Development*, 2005, 49(4): 555-564

[5] Ruan Y, Pai V S, Nahum E, et al. Evaluating the impact of simultaneous multithreading on network servers using real hardware. *ACM SIGMETRICS Performance Evaluation Review*, 2005, 33(1): 315-326

[6] Zaharia M, Chowdhury M, Das T, et al. Resilient distributed data sets: a fault-tolerant abstraction for in-memory cluster computing. In: Proceedings of the 9th Usenix Conference on Networked Systems Design and Implementation, San Jose, USA, 2012. 15-26

[7] Wang L, Zhan J, Luo C, et al. BigDataBench: a big data benchmark suite from internet services. In: Proceedings of the 20th IEEE International Symposium on High Performance Computer Architecture, Orlando, USA, 2014. 488-499

[8] Perf: Linux profiling with performance counters. https://perf. wiki. kernel. org/index. php/Main _ Page: Linux perf, 2015

[9] Ppc64 POWER8 events. http://oprofile. sourceforge. net/docs/ppc64-power8-events. php: Oprofile, 2014

[10] Apache Spark User List. http://apache-spark-user-list. 1001560. n3. nabble. com/executor-cores-vs-num-executors-td9878. html: Apache Spark, 2014

[11] Ferdman M, Adileh A, Kocberber O, et al. Clearing the clouds: a study of emerging scale-out workloads on modern hardware. In: Proceedings of the 17th International Conference on Architectural Support for Programming Languages and Operating Systems, London, UK, 2012. 37-48

[12] Jia Z, Wang L, Zhan J, et al. Characterizing data analysis workloads in data centers. In: Proceedings of 2013 IEEE International Symposium on Workload Characterization, Portland, USA, 2013. 66-76

[13] Bulpin J R, Pratt I A. Multiprogramming performance of the Pentium 4 with hyper-threading. In: Proceedings of the 2nd Annual Workshop on Duplicating, Deconstruction and Debunking, Munich, Germany, 2004. 53-62

[14] Zhang Y, Laurenzano M A, Mars J, et al. Smite: precise qos prediction on real-system SMT processors to improve utilization in warehouse scale computers. In: Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture, Cambridge, UK, 2014. 406-418

Jia Zhen, born in 1987. He is a Ph. D candidate in computer science at the Institute of Computing Technology, Chinese Academy of Sciences. He received his B. S. degree in 2006 from Dalian University of Technology in China. His research focuses on parallel and distributed systems, benchmarking and data center workload characterization.