國立交通大學

網路工程研究所

碩 士 論 文

為異質多核心上之爪哇虛擬機器設計一套模糊控制動態執行緒指定機制

Dynamic Thread Assignment with Fuzzy Control for Java Virtual Machine on Asymmetric Multicore Systems

研 究 生：邱筱惠

指導教授：楊武　教授

中 華 民 國　一百零一　年 六 月

為異質多核心上之爪哇虛擬機器設計一套模糊控制動態執行緒指定機制
Dynamic Thread Assignment with Fuzzy Control for Java Virtual Machine on Asymmetric Multicore Systems

研 究 生：邱筱惠　　Student：Hsiao-Hui Chiu

指導教授：楊武　　Advisor：Wuu Yang

國 立 交 通 大 學

網 路 工 程 研 究 所

碩 士 論 文

A Thesis

Submitted to Institute of Network Engineering

College of Computer Science

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of

Master

in

Computer Science

June 2012

Hsinchu, Taiwan, Republic of China

中華民國一百零一年六月

為異質多核心上之爪哇虛擬機器設計一套模糊控制動態執行緒指定機制

學生：邱筱惠　　　　　　　　　　　　指導教授：楊武 博士

國立交通大學網路工程研究所碩士班

摘　　　要

和同質多核心處理器比較起來，異質多核心處理器已經被提出作為一個比較好的折衷方案在於效能與能源消耗方面。在此同時也浮現了新的挑戰；那就是執行緒與核心之間的配對問題。 本論文提出一個基於模糊控制的排程器；在我們的研究中，我們架構了一個相同指令集架構但是不同核心有不同的頻率的異質多核心平台。我們所提出的模糊控制排程器週期性地收集執行緒的程式特性，並且根據這些特性來決定該執行緒應該被分配到哪一個核心上。實驗結果顯示，我們的模糊控制排程器對於記憶體存取密集的程式可以節省相當多能源以及得到顯著的能源使用效率而效能並未犧牲很多。

Dynamic Thread Assignment with Fuzzy Control for Java Virtual Machine on Asymmetric Multicore Systems

Student：Hsiao-Hui Chiu　　　　Advisor：Dr. Wuu Yang

Institute of Network Engineering
National Chiao Tung University

ABSTRACT

Asymmetric multicore processors have been proposed as a better trade-off between performance and power consumption over symmetric multicore processors. They also reveal challenges in mapping threads to cores. We propose a new scheduler based on fuzzy control theory. In this work, we configure an asymmetric multicore system in which cores share the same ISA but run at different frequencies. Our fuzzy scheduler decides thread-to-core assignment based on periodical run-time performance characteristics. Evaluation results demonstrate that our fuzzy scheduler saves significant energy and achieves better energy-delay product for memory-intensive programs while sacrificing performance slightly.

誌　　　　謝

能夠完成這篇論文，我最要感謝的是我的指導教授楊武老師，我很感謝老師的耐心指導與仁慈的心。在研究上，楊武老師給予我很多的自由去探索興趣。老師總是用幽默與智慧對待學生，讓我永生難忘。

感謝程式語言實驗室的眾多學長和學弟們，因為我讀了三年才認識那麼多人，這樣令我很慚愧。特別感謝柏曄學長與信慶，在研究上給我許多的幫助。

最後感謝我親愛的家人們，你們的鼓勵支持我走過最難熬的時期。最感謝我的父親，您的汗水足以淹沒一個成人。感謝您讓我在求學的時候毫無後顧之憂。以及我的母親和我親愛的妹妹貞蓉與小琪，在這些年中，你們的陪伴及歡笑，總能讓我暫時忘卻煩惱。

The work reported in this paper is partially supported by National Science Council, Taiwan, Republic of China, under grants NSC 100-2218-E-009-010-MY3 and NSC 100-2218-E-009-009-MY3.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Asymmetric Multicore Systems

Asymmetric multicore processors (AMPs) have been proposed as an alternative to provide a better trade-off between performance, power consumption and die-area over symmetric multicore processors (SMPs) [11, 16, 21]. There is a large diversity for designing AMPs, ranging from performance asymmetry to functional asymmetry [24]. Performance AMPs consist of several cores which differ in clock speeds, cache sizes, and microarchitectures [18,21], etc. Alternatively, functional AMPs allow cores to have different functionalities by employing different instruction set architectures (ISA). For example, AMPs may consist of CPU and GPU cores. Furthermore, cores may have overlapping ISAs, that is, some cores share a common ISA but have the special-purpose features [19, 20, 24]. We modelled the AMPs in which all

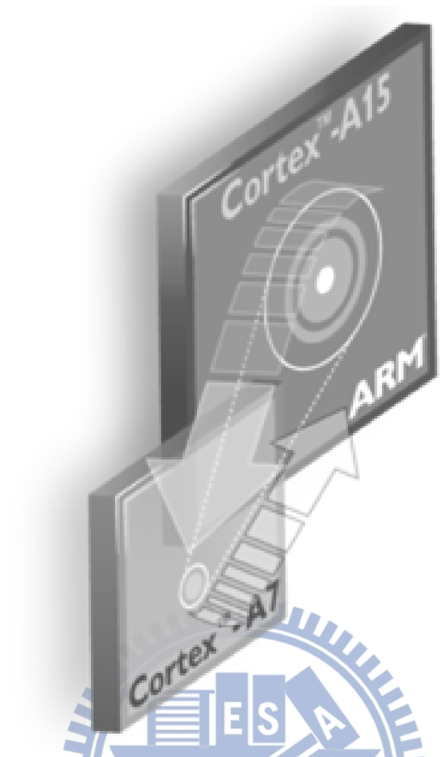


Figure 1.1: big.LITTLE Processing

cores support the same ISA but differ from different clock frequencies. Using the same ISA on all cores implies that the same object code may run all cores.

In October 2011, ARM announced a new chip architecture - big.LITTLE processing [12]. The goal of this approach to minimize power draw in order to extend the battery life of devices like smartphone and tablet. As shown in figure 1.1, big.LITTLE connects the performance of the ARM Cortex-A15 MPCoreTM processor with the energy efficiency of the Cortex-A7 processor, and enables the same application software to be seamlessly switched between them. By selecting the optimum processor for each task big.LITTLE can extend battery life by up to 70%. The big.LITTLE provides the opportunity to raise performance and extend battery life in the next generation of mobile platforms.

1.2 Toward Better Utilization of AMPs

AMPs reveal new software challenges. The main challenge is to utilize asymmetric multicores efficiently with a scheduler in the underlying operating system. For better utilization of asymmetric multicores, threads in a program should be assigned to cores such that the resource needs of each thread match the resource available at the assigned core. Many studies [6, 15, 17, 21, 25, 26] have proposed techniques to assign the threads to cores in order to achieve better compromise between performance and energy efficiency.

This paper presents a new mechanism based on fuzzy control [9, 31] to choose a particular core that matches the resource requirements of an application thread to maximize power efficiency within AMPs. Our mechanism is implemented in JVM. This JVM, which is called *asymmetry-aware JVM*, is aware of the asymmetric hardware properties of the system and assigns threads to cores based on thread characteristics. The thread characteristics of each thread are obtained from hardware performance counters periodically and are used as input to the fuzzy-control scheduler.

The intuition behind our fuzzy control scheduler is as follows. Faster and more powerful cores are good for running CPU-intensive programs which are able to exploit the advanced features of processors. On the other side, slower and simpler cores save energy for memory-intensive programs, which spend most of the execution time on fetching data from memory and stalling CPU pipelines [10]. In the fuzzy-control scheduler, we profile each executing thread in JVM by hardware performance counters and classify it as CPU-intensive or memory-intensive depending on the performance metrics. An appropriate core is selected for each Java thread.

Choosing appropriate performance metrics for the fuzzy-control scheduler is an important issue. In our previous work [27], we chose the instructions per cycle (IPC) of the executing thread because IPC strongly correlates to throughput and utilization of processors. Unfortunately, IPC alone is not sufficient to make a right core decision because not all programs with lower IPC were memory-intensive. There are other reasons for lower IPC [7]. Thus, other performance metrics become more important, such as cache behavior, branch predictor behavior, and utilization of resources. In this work, we select several additional performance metrics to make more accurate core decision. We will describe the selected metrics in Chapter 3.

Chapter 2

Related work

Many studies address the scheduling problems for asymmetric multicore systems. Some of these studies are static scheduling algorithms and others are dynamic.

In [25], Shelepov et al. proposed an asymmetry-aware scheduler by using offline-generated architectural signatures for assigning threads to cores in AMPs. The static way does not consider the runtime characteristics of application; the resource requirement of application may vary across while executing. Thus, the dynamic approach takes the runtime characteristics into account.

In [15, 17], Kumar et al. proposed a sampling-based dynamic scheduler. The scheduler consisted of two phases, a sampling phase and a steady-state phase. In the sampling phase, the scheduler chose a proper core based on profiling statistics of the application, which was gathered by hardware performance counters. The scheduler switched the application to the selected core. Their work showed that asymmetric multicore systems can reduce significant energy consumption with little sacrifice in performance. Our work adopts the dynamic-sampling approach with fuzzy control to choose a core for each thread.

In [6], Becchi and Crowley proposed work similar to that of Kumar. They defined new dynamic assignment policies, including round-robin and IPC-driven. In particular, the IPC-driven assignment depended on the ratio between two different core types to make core decision.

Some of schedulers are phase-based [22, 26, 29, 30]. One of phased-based schedulers is proposed by Sondag and Rajan. Their approach [26] is composed by two subsections, phase detection and core assignment. In the phase detection, a static detection algorithm was used to identify phase-transition point. The static analysis divided a complete program into several sections and classified them into several groups, in which all the sections are likely to exhibit similar runtime characteristics. In the core assignment subsection, the dynamic analysis collected the runtime behaviors of a section of each phase type. The runtime characteristics are representative to each phase type. These characteristics are used to choose a suitable core that matched the resource requirement for each phase type. This kind of scheduler is more complicated than sampling based schedulers because the designer needs to implement both static and dynamic analysis.

Fuzzy control [9, 31] is famous in control theory. Previous study [23] showed the DVFS decisions made by fuzzy control depended on variation of workloads. Their work inspired us to design the core assignment by fuzzy control theory. In our prior work [27], we proposed two sampling-based fuzzy schedulers to achieve the performance and energy efficiency on AMPs: IPC fuzzy scheduler and LLC-misses fuzzy scheduler. In the IPC fuzzy scheduler, we use the IPC value and the IPC difference as input to the fuzzy logic block. The reason of using IPC is that IPC strongly correlates to the throughput and utilization of processors. In considering the factors causing performance getting worse, the last-level-cache (LLC) miss is the main factor. Thus, we proposed the LLC-misses fuzzy scheduler, in which we use the number of LLC-load-misses and the LLC miss difference as input to the fuzzy logic block. The result showed that these fuzzy schedulers could make a right decision for memory-intensive programs, but may make a wrong decision for some non-memory-intensive programs. In our work, we consider other performance metrics as input to the fuzzy logic to make a right decision. We evaluate the performance and energy efficiency of our new fuzzy-control schedulers.

Chapter 3

Implementation

3.1 JVM and JVM Tool Interface

Java Virtual Machine is a good environment to profile program behaviors of Java applications, because the VM has JVMTI [2], a convenient tool was introduced in J2SE 5.0 and completely independent of specific JVM implementation. JVMTI provides an interface to inspect the execution of Java applications. In our work, we implemented an agent by JVMTI to track each Java thread in JVM on the fly. Note that in Hotspot JVM, the mapping between Java threads and Linux threads is implemented by Pthreads library which creates a kernel thread for each user thread, and is known as one-to-one mapping. That means we can monitor each Java thread by Linux Performance Counter Subsystem [3], which uses to count the number of certain types of hardware events for processes or threads. It provides a system call *sys_perf_event_open* for hardware performance counter. The system call returns a file descriptor for each hardware performance counter event we monitored. We can read those files to get the number of counters.

3.2 Interesting Performance Metrics

IPC alone is not an accurate indicator of performance [7]. When IPC value is lower, other performance metrics become more important. There are several reasons causing lower IPC value. By examining several programs, we found out that cache misses and data dependency are the main reasons, so we made core decision by considering cache misses and resource stall cycles in our work. We choose the following performance metrics as inputs of fuzzy scheduler:

- **Resource stalls ratio (RSR)** which represents several kinds of potential performance stalls including frequent cache misses, long execution paths, and memory order buffer (MOB) stalls.
- **L2 and LLC cache miss rate** which are used to know how heavy the cache misses effect the performance of application. The L2 cache miss rate and LLC miss rate are measured as misses per 1000 instructions in our design, then we named the L2 MPI and LLC MPI which representing for L2 and LLC misses per a thousand instructions [13].

We obtained the following performance events in Intel i7 processor [14] to calculate the RSR, L2 and LLC MPI:

- `RESOURCE_STALLS.ANY`: This event counts the number of cycles while resource-related stalls occur, including the number of instructions in the pipeline waiting for execution and there is an instruction in the pipeline that can be executed only when all previous stores complete and their data is committed in the caches or memory.
- `CPU_CLK_UNHALTED.THREAD_P`: This event counts the number of thread cycle while the thread is not in halt state.
- `INST_RETIRED.ANY_P`: This event counts the number of instructions that retired from execution.
- `L2_RQSTS.MISS`: This event counts all L2 misses for both code and data.
- `LLC MISSES`: This is an architectural performance event which counts the number of last level cache misses.

3.3 Thread Assignment Policy

The heuristic of thread assignment policy in our scheduler is as following. We do not need to make core decision for those threads with higher IPC value($ipc \geq 1.5$) to reduce the overhead of fuzzy control calculation, because this situation is definitely scheduled to the fastest cores by our scheduling. Choosing 1.5 as a threshold of doing fuzzy scheduling depends on benchmark evaluation. The main goal of our scheduler is making proper core decision for those threads with lower IPC value by considering the cache behaviors and resource stall rates during runtime. This heuristic was implemented with

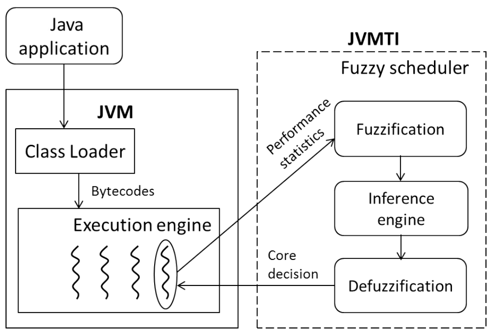


Figure 3.1: Architecutre of the fuzzy scheduler using JVMTI

fuzzy control, and we designed a fuzzy scheduler, which did twice fuzzification processing.

In the first fuzzification, we considered the RSR and L2 cache miss rate. The purpose of the first fuzzification is to recognize whether data dependency is the main cause of higher stall rate or not. If so, it means that this thread is not memory-intensive and should be scheduled to faster cores. Otherwise, higher cache miss rate are the main cause of higher stall rate.

In the second fuzzification, we considered the LLC miss rate to examine how heavy the cache behaviors affect the performance. If LLC miss rate are higher, this thread is memory-intensive and should be scheduled to slower cores.

3.4 Fuzzy Scheduler

The fuzzy scheduler which is implemented for our asymmetry-aware JVM is a closed loop which takes profiling statistic as input and makes core decision

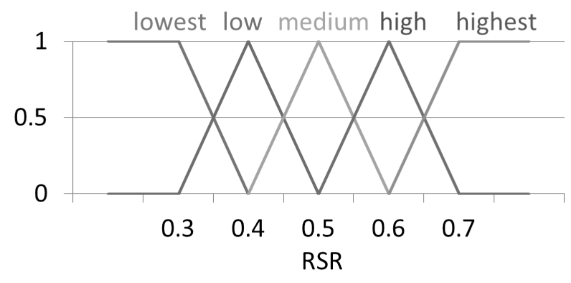


Figure 3.2: Membership functions for RSR

as output periodically. Figure 3.1 presents our scheduler. As it is showed in figure 3.1, the fuzzy scheduler consists of three blocks: a fuzzification block, inference engine block and defuzzification block. The fuzzification block converts a quantitative value into a qualitative value. In this block, membership functions should be considered for each control state. Membership function is a function that specifies the degree for each input which belongs to a fuzzy set, and its value is limited between 0 and 1. The second block is inference engine which is using If-Then fuzzy rules to infer the input fuzzy set to the output fuzzy set. The third part is defuzzifiation block which converts the output fuzzy set to a crisp number that can be used as a control signal.

In the fuzzification part of our scheduler, there are several linguistic variables; each of them stands for a concept and is associated with one or more membership functions. We declared two linguistic variables in the first fuzzification: RSR and L2 MPI which are as mentioned previously. We also defined five membership functions for RSR and four for L2 MPI as showing in figure 3.2 and figure 3.3. Although there are many possible shapes of each

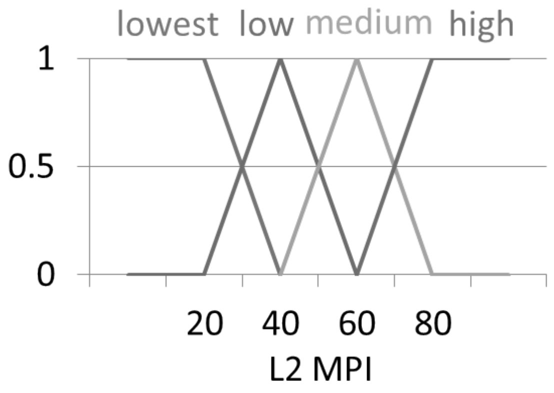


Figure 3.3: Membership functions for L2 MPI

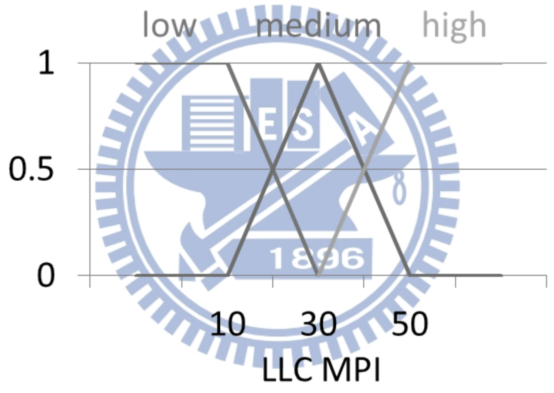


Figure 3.4: Membership functions for LLC MPI

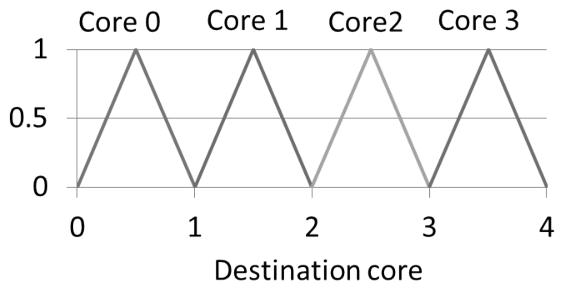


Figure 3.5: Membership functions for destination core in asymA

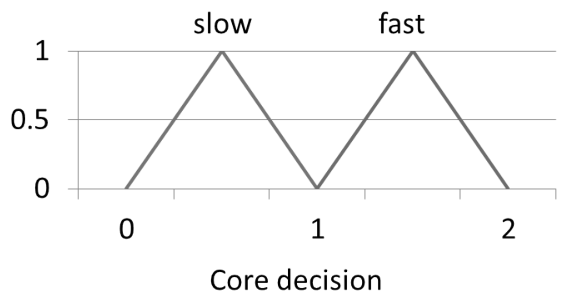
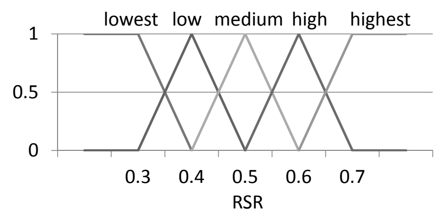
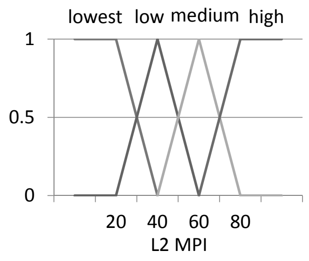


Figure 3.6: Membership functions for destination core in asymB

membership function, the shape does not affect the output significantly [23]. For this reason and to minimize computational complexity, we choose triangular membership function in our case. The overlap between two membership functions is 50%. That means each RSR input or L2 MPI input will hit at most two membership functions and at most four rules to do the inference. For example, we have a set of performance statistics, which RSR value is 0.67 and L2 MPI is 12. RSR will hit high and highest membership functions in figure 3.2, and L2 MPI will hit low and lowest in figure 3.3. In the first fuzzification inference engine, some of rules will get into the second fuzzfication and then return the output value to first fuzzification to do the rest of processing. For example, if RSR is highest and L2 MPI is low, this rule will get into the second fuzzfication. In the second fuzzification, we considered the cache behavior. We had L2 MPI and LLC MPI as linguistic variables and defined three new membership functions for LLC MPI as showing in figure 3.4. The shape of membership functions is similar with those in the first fuzzification.

First fuzzification:



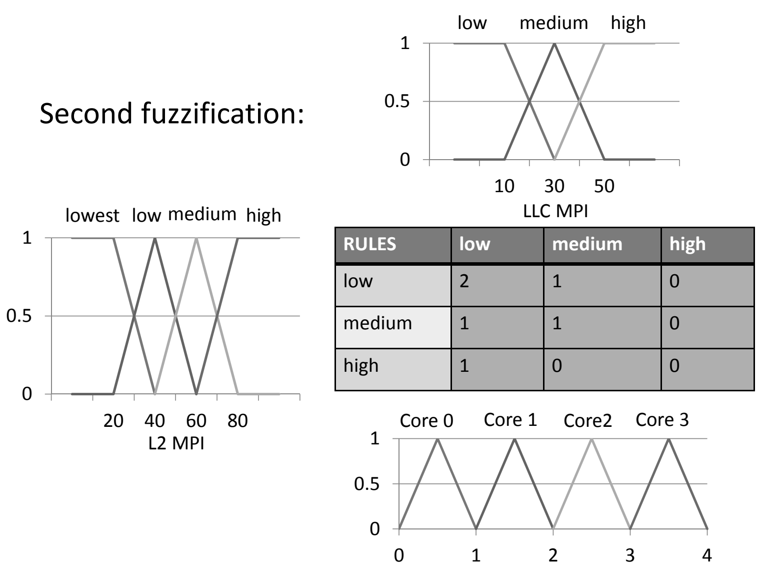


RULES	lowest	low	medium	high	highest
lowest	2	2	3	3	3
low	F2	F2	F2	F2	F2
medium	F2	F2	F2	F2	F2
high	F2	F2	F2	F2	F2

Figure 3.7: Inference rules for RSR and L2 MPI

The design of the output membership functions highly depends on the target AMPs. Figure 3.5 shows the membership functions for the output variable destination core.There are four cores with different frequencies in our first asymmetric configuration, so we defined four membership functions for each core. Besides, we also configured AsymB, the other asymmetric configuration with two kinds of cores, so we designed two membership functions as shown in figure 3.6.

The inference engine consists of several rules, and these rules are defined based on our background knowledge and observation. Since we have five membership functions for RSR variable and four for L2 MPI variable in the first fuzzification, there should be 20 rules. Similarly, four for L2 MPI variable



RULES	low	medium	high
low	2	1	0
medium	1	1	0
high	1	0	0

Figure 3.8: Inference rules for L2 MPI and LLC MPI

and three for LLC MPI variable, so there should be 12 rules in the second fuzzification. Figure 3.7 and figure 3.8 showed the rules in two fuzzification steps. The inference will produce implied membership functions which will be used to calculate a crisp value in defuzzification block.

The defuzzification part converts the implied fuzzy set to a crisp value as a control signal. There are several defuzzification methods such as center of gravity, center of area, mean of maximum, and first of maximum. We choose center of area as our defuzzification method. Then crisp value will be a core decision which the scheduling thread assigns the running thread to selected core.

Chapter 4

Experiment Setup

4.1 System Configuration

We use an Intel core i7-920 processor as our hardware platform which consists of an AMP with four cores and the detailed system specification is shown in table1. Note that both Intel Turbo Boost Technology and Intel Hyper-Threading Technology are disabled to eliminate their effect on performance and power consumption. It is worth to mention that the L3(LLC) cache in our experiment platform is shared. According to previous research [28], the overhead of thread migration is negligible. So our work would not suffer from significant performance loss as running on our platform.

We use an unmodified Linux 2.6.39 kernel. One of Linux kernel modules acpi-cpufreq is used to scale the CPU speed up or down to save power. We use this module to configure our asymmetric environment. Table2 shows the

Table 4.1: intel i7-920 specification

Platform	Intel i7-920
# of cores	4
Max. & Min. speed	2.66GHz & 1.6GHz
L1 Cache	32KB L1 data, 32KB L1 instruction per core
L2 Cache	256KB per core, inclusive
L3 Cache	8 MB shared cache
Memory	8GB
OS	Debian-amd64 (kernel 2.6.39-2)
Java version	1.6.0_18
OpenJDK	IcedTea6 1.8.7

Table 4.2: Available performance choice (GHz)

1.60	1.73	1.86	2.00	2.13	2.26	2.39	2.53	2.66

four configurations in our experiments. The symmetric configuration sym2.66 consists of four cores with highest frequency 2.66GHz and is used to be as the baseline. The other symmetric configuration sym2.26 is used to show the worst case. The asymmetric configuration AsymA consists of four cores with fastest four frequencies. The other asymmetric configuration AsymB consists of two kinds of core, fast and slow. AsymB is with less diversity but more asymmetry than AsymA, and this is used to compare the performance and power consumption between different asymmetric configurations.

Table 4.3: 4 configurations in our experiment

	CPU 0	CPU 1	CPU 2	CPU 3
Sym2.66	2.66GHz	2.66GHz	2.66GHz	2.66GHz
Sym2.26	2.26GHz	2.26GHz	2.26GHz	2.26GHz
Asym A	2.26GHz	2.39GHz	2.53GHz	2.66GHz
Asym B	2.00GHz	2.00GHz	2.66GHz	2.66GHz

4.2 Benchmark Configuration

There are two suits of benchmark in our experiments: Scimark and Java Grande Forum (JGF) benchmarks [8]. The reason why we choose these benchmark applications is that they demand on a lot of memory or computing resources. The Scimark applications are from SPECjvm2008 [5], and we configured them with `lagom` which is a option to fix the amount of operations. An exception is scimark.monte_carlo application. Because its lagom is with 900 operations, we reduced them to 100 operations for execution efficiency. The Scimark workloads are run with both small and large data sizes. The large is 32MB and the small is 512KB. In the JGF benchmarks, there are three sections. We chose section II and section III for large scale computation. Unlike Scimark, the data sizes of JGF benchmark applications are not same as each other. There are three kinds of data sizes for sectionII and two for sectionIII. We chose the largest data size in our experiments. Note that the number of threads of each application was set to one to meet the constraints of our power meter.

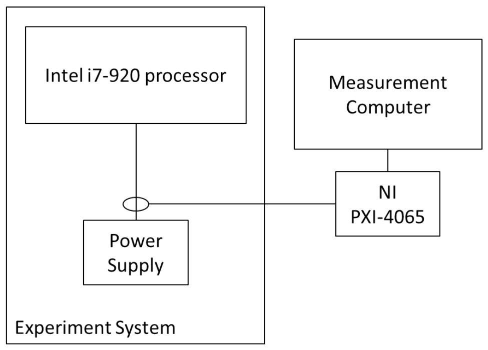


Figure 4.1: Power Measurement Setup

4.3 Power and Energy Measurement

The power of Intel i7 processor is provided by 12V2 rails, which is according to ESP12V power supply design guide [1]. We made an instrument between power supply and power meter. The diagram of the power measurement setup can be seen in figure 4.1. We use NI PXI-4065 digital multimeter to measure the power dissipation. The current trace was interpretered by the LabVIEW software and used to compute power consumption. Project SIKULI [4] is used to automate the process of measurement.

Chapter 5

Evaluation

5.1 Benchmark Characteristics

To obtain the baseline of our experiments, we first performed a series of measurements to characterize the individual benchmark applications. We ran each benchmark application at the fastest cores with frequency 2.66GHz as baseline, which completes the application in the least amount of time, but gains no energy saving. We also ran benchmark applications on other lower frequencies ranging from 1.60GHz to 2.52GHz. To measure and compare the power efficiency, we choose the energy delay product (EDP) as the power efficiency matric. The EDP value for each benchmark was normalized to the value at 2.66GHz and showed in figure 5.1 and figure 5.2.

The fft benchmark in scimark.*.large and JGF suite, experiences more EDP benefit while running at lower frequencies. In contrast, all other bench-

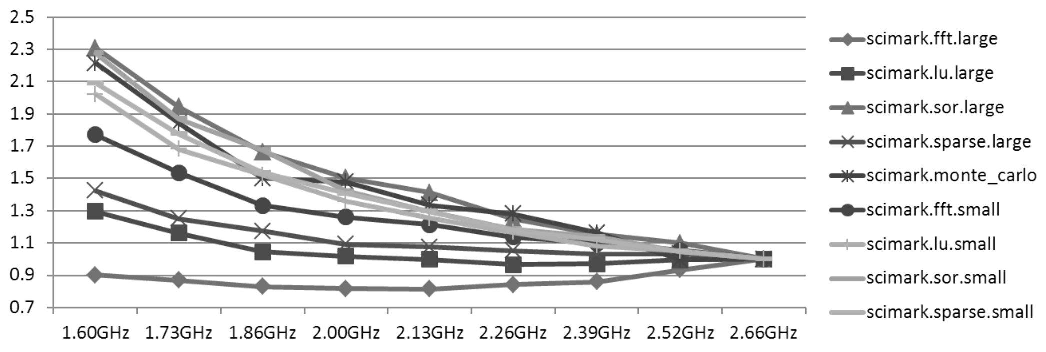


Figure 5.1: Relative EDP of scimark programs in SPECjvm2008

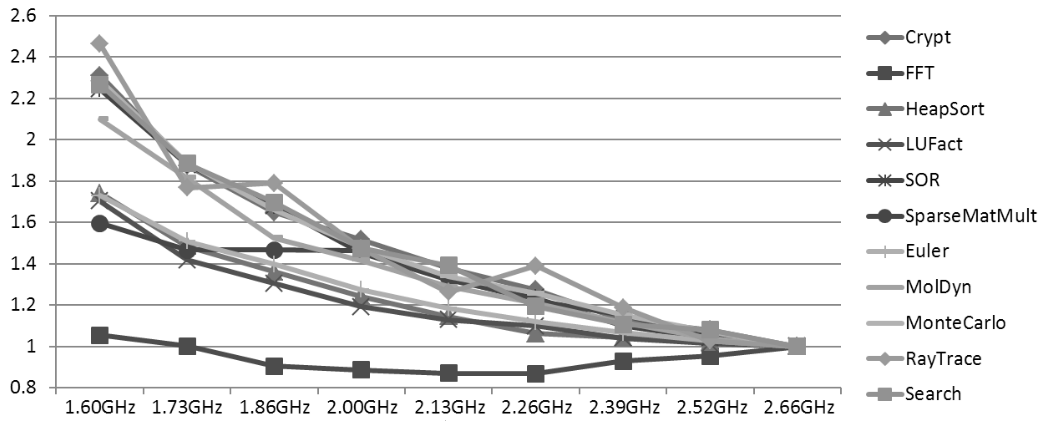


Figure 5.2: Relative EDP of JGF programs

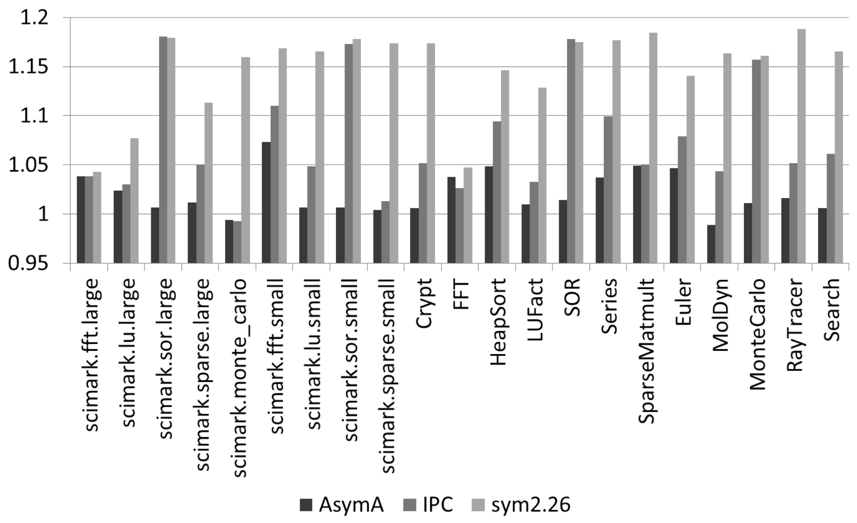


Figure 5.3: Execution time ratio for Fuzzy scheduler and IPC scheduler in asymA

marks suffer more performance loss than energy saving, so they could not get any EDP benefit. The fft program is obviously less CPU-intensive than others and spends more time waiting for memory access. Thus, scheduling this kind of programs to slower cores affects its execution time less than execution time of other programs, but can get more energy saving than others.

5.2 Experiment Results

In the first part of our experiments, we compare the results of our fuzzy scheduler to the results of IPC scheduler in asymmetric configuration A. Note that all the results are normalized to the results at 2.6Ghz. Figure 5.3 gives the execution time ratio for our fuzzy scheduler and IPC scheduler. We also

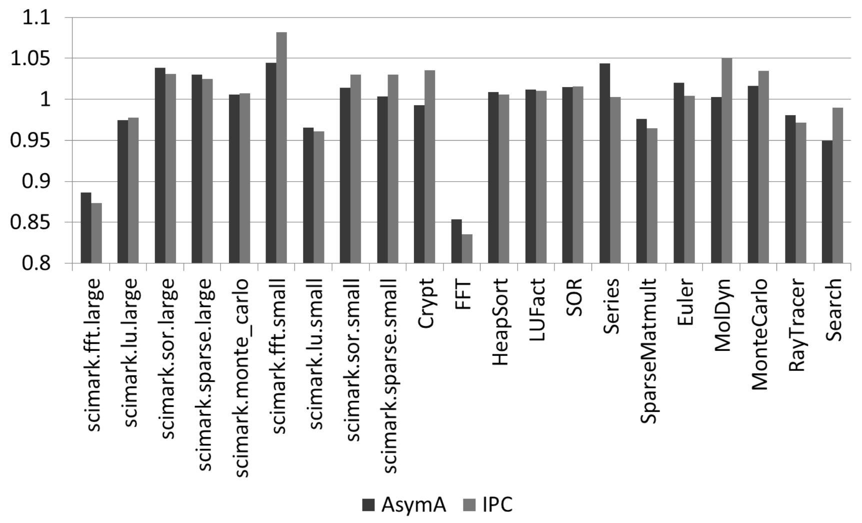


Figure 5.4: Relative EDP for Fuzzy scheduler and IPC scheduler in asymA

provide the execution ratio in symmetric configuration sym2.26, as 2.26GHz is the lowest frequency in asymA. This gives a comparison to the worst time increasing in our case. Figure 5.3 shows that the fuzzy scheduler is able to make right decision for each benchmark program, especially for scimark.sor.*, SOR and MonteCarlo. These programs are CPU-intensive and should be scheduled to the fastest core. Unfortunately, the IPC scheduler made wrong core decision for them so that it resulted in the worst time increasing. All benchmark programs slightly suffer from 2% performance loss.

The execution time ratio in sym2.26 indicates that scheduling scimark.fft.large, scimark.lu.large and FFT to the slowest core did not cause significant performance loss, especially for scimark.fft.large and FFT which are memory-intensive. Thus, these programs could gain more EDP benefit than others potentially.

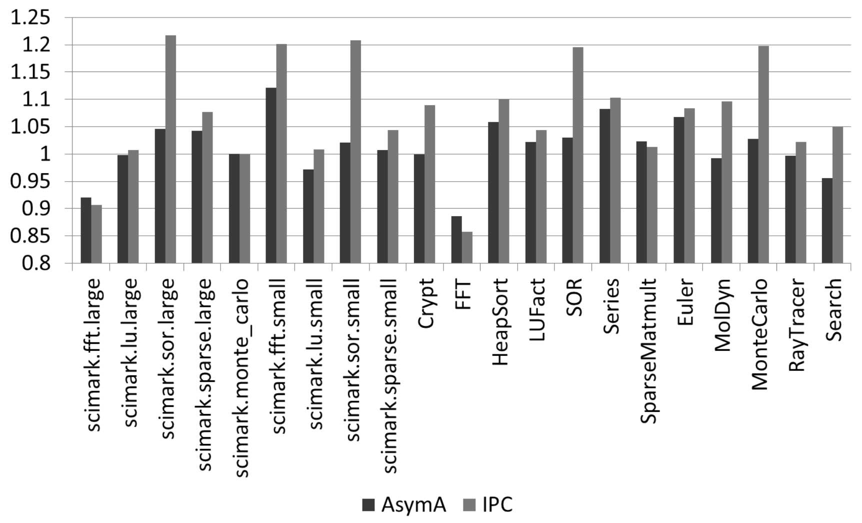


Figure 5.5: Relative EDP for Fuzzy scheduler and IPC scheduler in asymA

Figure 5.4 shows relative EDP for the fuzzy scheduler and IPC scheduler in asymmetric configuration A. Even though the fuzzy scheduler made right core decision for all the programs, they still suffered from EDP increase slightly. In figure 5.4, scimark.fft.large and FFT got 13% EDP benefit in average, because the two programs are known as memory-intensive. The relative EDP of scimark.sor.* and SOR for two schedulers shows that the compromise between performance and power saving is almost same to each other. That means even if we made right decision for sor series programs, we did not gain any EDP benefit. But if we choose ED^2P as the power efficiency matric which emphasizes performance more than power dissipation, we can get ED^2P benefit as shown in Figure 5.5.

Figure 5.6 gives the core decision for the fuzzy scheduler in asymA. For most of benchmark programs were scheduled to the fastest core besides

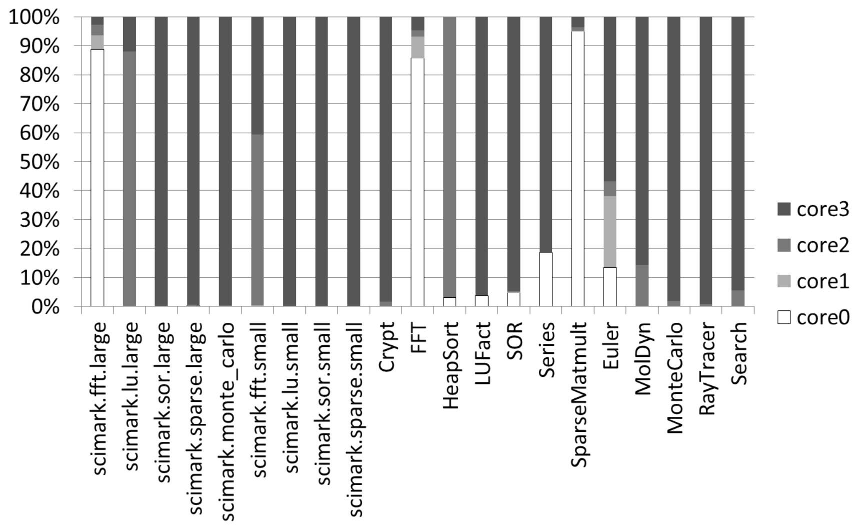


Figure 5.6: Core decision for Fuzzy scheduler in asymA

memory-intensive programs. Thus, we configured another asymmetric configuration which is less diversity but more asymmetry than asymA to get more power saving and EDP benefit.

In the second part of our experiments, we compare the performance and power consumption between different asymmetric configurations. As shown in table 4.3, asymB consists of two kinds of cores, fast cores with frequency 2.66GHz and slow cores with frequency 2.00GHz. In the following paragraphs, we will compare the execution time, power consumption, and EDP between asymmetric configuration A and B.

Figure 5.7 shows that execution time of most of programs are almost same as time execution time in asymA, because these programs were scheduled to the fastest cores with 2.66GHz as same as the fastest frequency in asymA. For memory-intensive programs such as scimark.fft.large and FFT, the execution

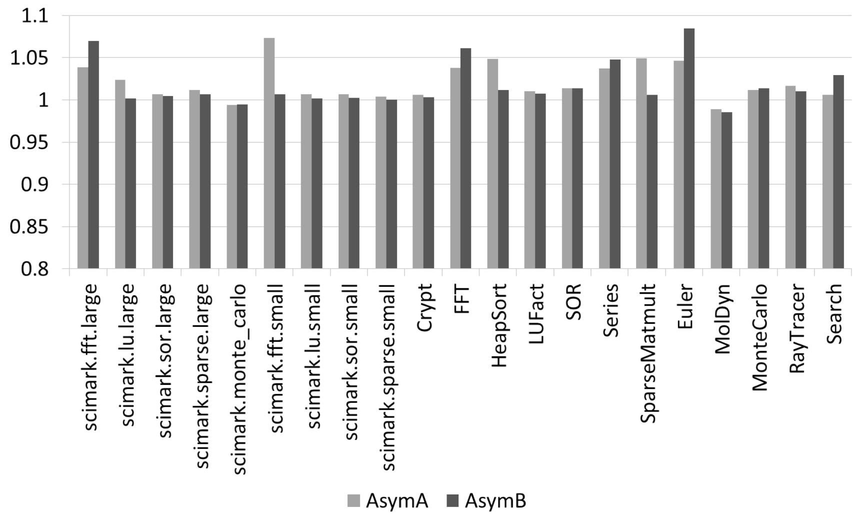


Figure 5.7: Execution time ratio for Fuzzy scheduler in asymA and asymB

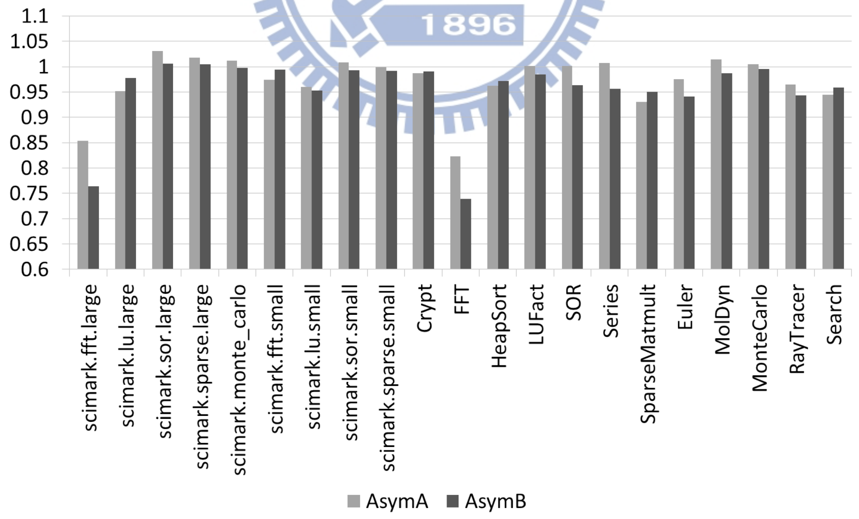


Figure 5.8: Power consumption ratio for Fuzzy scheduler in asymA and asymB

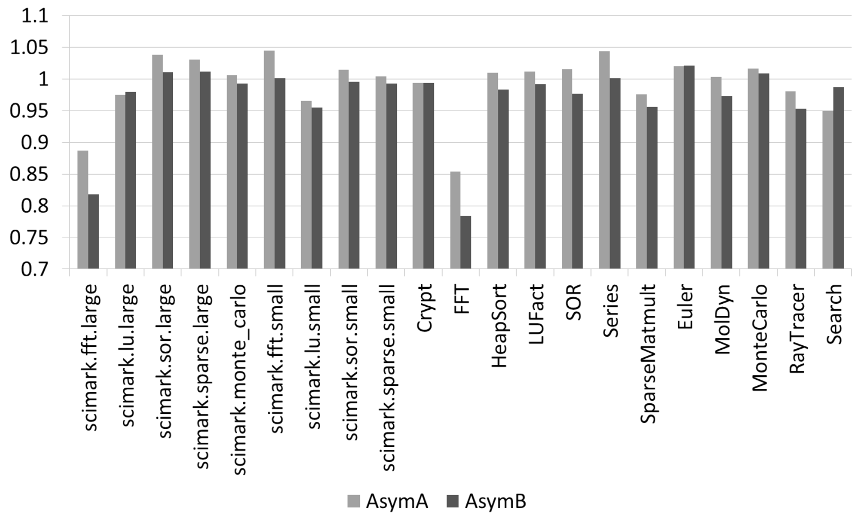


Figure 5.9: Relative EDP for Fuzzy scheduler in asymA in asymB

time is increasing slightly at 4%, but save 25% power in average as shown in figure 5.8. It means it can make a better compromise between perofrmance and power efficiency compared to asymA. Figure 5.9 shows that we got 20% EDP benefit and 7% more than it in asymA.

Figure 5.10 gives the core decision for the fuzzy scheduler in asymB. Only the memory-intensive programs were scheduled to the slow cores. One of advantages in asymB is that it reduced thread migration times. For some of CPU-intensive programs, there is no thread migration at all.

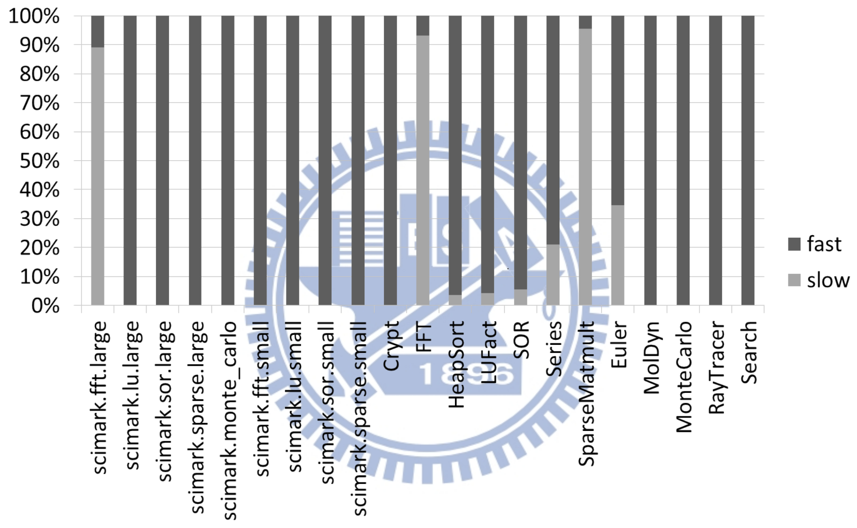


Figure 5.10: Core decision for Fuzzy scheduler in asymB

Chapter 6

Conclusions and Future works

6.1 Conclusions

Asymmetric multicores processors have been recently proposed a good compromise between performance and power efficiency. However, they also face new challenges such as scheduling problems and resource utilization problems. Many studies devised new technique to utilize AMPs efficiently. [6, 15, 18, 26]. This thesis presents a fuzzy scheduler based on fuzzy control theory for JVM to migrate Java threads to the proper cores for energy saving on AMPs. In our previous work, the IPC scheduler could not make a right core decision for some of CPU-intensive programs because it did not consider data dependency and that would cause execution time to increase heavily. Our fuzzy scheduler takes RSR and cache miss rate into account such that the fuzzy scheduler could make the right core decision for all benchmark programs. Our experiments show 25% energy saving and 20% EDP benefit for memory-intensive applications compared to the results at the fastest core. In our experiments, we could not get any benefit for CPU-intensive programs, because they could not execute at a lower frequency efficiently.

6.2 Future works

In the future work, we would like to consider more about OS scheduling issues, such as load balance and asymmetry-aware OS thread assignment. We would also like to eliminate the constraint of thread number to make more efficient utilization. One of other future directions is to use a wider diversity of asymmetric multicore systems in different cache sizes, or having different number of execution units, or supporting in-order or out-of-order execution and so on.

Bibliography

[1] *EPS12V Power Supply Design Guide - A Server System Infrastructure (SSI) Specification For Entry Chassis Power Supplies*, 2.91 edition.

[2] Jvm tool interface.

[3] Performance counters on linux.

[4] Project sikuli.

[5] Specjvm2008.

[6] Michela Becchi and Patrick Crowley. Dynamic thread assignment on heterogeneous multiprocessor architectures. In *Proceedings of the 3rd conference on Computing frontiers*, CF '06, pages 29–40, New York, NY, USA, 2006. ACM.

[7] W. L. Bircher, Jason Law, Madhavi Valluri, and Lizy K. John. Effective use of performance monitoring counters for run-time prediction of power. Technical report, nov. 2004.

[8] J. Mark Bull, L. A. Smith, M. D. Westhead, D. S. Henty, and R. A. Davey. A benchmark suite for high performance java. *Concurrency - Practice and Experience*, pages 375–388, 2000.

[9] Didier Dubois. Fuzzy sets and their applications : Vilem novak, translated from czechoslovakian. bristol and philadelphia: Adam hilger, 1989, 248 pages. *Mathematical Social Sciences*, 21(2):193–197, April 1991.

[10] Alexandra Fedorova, Juan Carlos Saez, Daniel Shelepov, and Manuel Prieto. Maximizing power efficiency with asymmetric multicore systems. *Commun. ACM*, 52(12):48–57, December 2009.

[11] Matt Gillespie. Preparing for the second stage of multi-core hardware: Asymmetric (heterogeneous) cores. Technical report, Intel, 2008.

[12] Peter Greenhalgh. Big.little processing with arm cortex-a15 & cortex-a7. White paper, ARM, 2011.

[13] John Hennessy, John L. Hennessy, David Goldberg, and David A. Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann Publishers, 4rd edition.

[14] Intel Corporation. *Intel 64 and IA-32 Architectures Software Developer's Manual - Volume 3B: System Programming Guide*, May 2011. chapter 30 Performance monitoring.

[15] R. Kumar, K.I. Farkas, N.P. Jouppi, P. Ranganathan, and D.M. Tullsen. Single-isa heterogeneous multi-core architectures: the potential for pro-

cessor power reduction. In *Microarchitecture, 2003. MICRO-36. Proceedings. 36th Annual IEEE/ACM International Symposium on*, pages 81 – 92, dec. 2003.

[16] R. Kumar, D.M. Tullsen, N.P. Jouppi, and P. Ranganathan. Heterogeneous chip multiprocessors. *Computer*, 38(11):32 – 38, nov. 2005.

[17] Rakesh Kumar, Dean M. Tullsen, Parthasarathy Ranganathan, Norman P. Jouppi, and Keith I. Farkas. Single-isa heterogeneous multi-core architectures for multithreaded workload performance. *SIGARCH Comput. Archit. News*, 32(2):64–, March 2004.

[18] Tong Li, Dan Baumberger, David A. Koufaty, and Scott Hahn. Efficient operating system scheduling for performance-asymmetric multicore architectures. In *Proceedings of the 2007 ACM/IEEE conference on Supercomputing*, SC '07, pages 53:1–53:11, New York, NY, USA, 2007. ACM.

[19] Tong Li, P. Brett, R. Knauerhase, D. Koufaty, D. Reddy, and S. Hahn. Operating system support for overlapping-isa heterogeneous multi-core architectures. In *High Performance Computer Architecture (HPCA), 2010 IEEE 16th International Symposium on*, pages 1 –12, jan. 2010.

[20] Tong Li, Paul Brett, Barbara Hohlt, Rob Knauerhase, Sean D. McElderry, and Scott Hahn. Operating system support for shared-isa asymmetric multi-core architectures. In *Proceedings of the Fourth*

Annual Workshop on the Interaction between Operating Systems and Computer Architecture (WIOSCA '08), pages 19–26, June 2008.

[21] J.C. Mogul, J. Mudigonda, N. Binkert, P. Ranganathan, and V. Talwar. Using asymmetric single-isa cmps to save energy on operating systems. *Micro, IEEE*, 28(3):26 –41, may-june 2008.

[22] Priya Nagpurkar, Chandra Krintz, Michael Hind, Peter F. Sweeney, and V. T. Rajan. Online phase detection algorithms. In *Proceedings of the International Symposium on Code Generation and Optimization*, CGO '06, pages 111–123, Washington, DC, USA, 2006. IEEE Computer Society.

[23] H.R. Pourshaghaghi and J.P. de Gyvez. Dynamic voltage scaling based on supply current tracking using fuzzy logic controller. In *Electronics, Circuits, and Systems, 2009. ICECS 2009. 16th IEEE International Conference on*, pages 779 –782, dec. 2009.

[24] Dheeraj Reddy, David Koufaty, Paul Brett, and Scott Hahn. Bridging functional heterogeneity in multicore architectures. *SIGOPS Oper. Syst. Rev.*, 45(1):21–33, February 2011.

[25] Daniel Shelepov, Juan Carlos Saez Alcaide, Stacey Jeffery, Alexandra Fedorova, Nestor Perez, Zhi Feng Huang, Sergey Blagodurov, and Viren Kumar. Hass: A scheduler for heterogeneous multicore systems. *SIGOPS Oper. Syst. Rev.*, 43:66–75, April 2009.

[26] T. Sondag and H. Rajan. Phase-based tuning for better utilization of performance-asymmetric multicore processors. In *Code Generation and Optimization (CGO), 2011 9th Annual IEEE/ACM International Symposium on*, pages 11 –20, april 2011.

[27] Hsin-Ching Sun, Bor-Yeh Shen, Wuu Yang, and Jenq-Kuen Lee. Migrating java threads with fuzzy control on asymmetric multicore systems for better energy delay product. In *International Conference on Computing and Security*, Ulaanbaatar, Mongolia, July 2011.

[28] Qiming Teng, P.F. Sweeney, and E. Duesterwald. Understanding the cost of thread migration for multi-threaded java applications running on a multicore platform. In *Performance Analysis of Systems and Software, 2009. ISPASS 2009. IEEE International Symposium on*, pages 123 –132, april 2009.

[29] Viswanath Krishnamurthy Tyler Sondag and Hridesh Rajan. Predictive thread-to-core assignment on a heterogeneous multi-core processor. In *PLOS '07: ACM SIGOPS 4th Workshop on Programming Languages and Operating Systems*, October 2007.

[30] Frederik Vandeputte, Lieven Eeckhout, and Koen De Bosschere. Exploiting program phase behavior for energy reduction on multi-configuration processors. *J. Syst. Archit.*, 53(8):489–500, August 2007.

[31] Lotfi A. Zadeh. Fuzzy logic, neural networks, and soft computing. *Commun. ACM*, 37(3):77–84, March 1994.