國立交通大學

資訊科學與工程研究所

碩 士 論 文

在異質多核心系統上利用模糊控制管理爪哇執行緒以達到更好的能源使用效率

Migrating Java Threads with Fuzzy Control on Asymmetric Multicore Systems for Better Energy Delay Product

研 究 生：孫信慶

指導教授：楊武　教授

中 華 民 國 一 百 年 六 月

在異質多核心系統上利用模糊控制管理爪哇執行緒以達到更好的能源使用效率
Migrating Java Threads with Fuzzy Control on Asymmetric Multicore Systems for Better Energy Delay Product

研 究 生：孫信慶　　Student：Hsin-Ching Sun

指導教授：楊武 博士　　Advisor：Dr. Wuu Yang

國 立 交 通 大 學
資 訊 科 學 與 工 程 研 究 所
碩 士 論 文

A Thesis
Submitted to Department of Computer Science and Engineering
College of Computer Science
National Chiao Tung University
in partial Fulfillment of the Requirements
for the Degree of
Master
in

Computer Science and Engineering

June 2011

Hsinchu, Taiwan, Republic of China

中華民國 100 年 6 月

在異質多核心系統上利用模糊控制管理爪哇執行緒以達到更好的能源使用效率

學生：孫信慶　　　　　　　　　　指導教授：楊武 博士

國立交通大學資訊科學與工程研究所碩士班

摘　　要

為了增進應用程式執行時，性能與能源消耗的效率，學者們提出了新的硬體平台—異質多核心系統。在異質多核心系統中，每一顆核心可以擁有不同的性能與功率。要運用異質多核心系統時，一個主要的問題是如何分配不同的核心資源給應用程式。在這篇論文當中，我們利用動態電壓頻率調整技術建立了一個虛擬的異質多核心系統，目的是為了得到物理上的功率測量資料。我們為一般的爪哇虛擬機器設計了一個執行緒管理員，這個執行緒的管理員將針對爪哇執行緒，利用硬體效能計數物件的資料實施模糊控制管理，目標是更好的能源使用效率。我們利用能源與執行時間的乘積作為能源使用效率的指標。主要的實驗結果包括:實施模糊控制管理的爪哇虛擬機器，在 scimark.fft.large 上取得能源使用效率的好處。並且，整體能源的使用是下降的。

Migrating Java Threads with Fuzzy Control on Asymmetric Multicore Systems for Better Energy Delay Product

Student：Hsin-Ching Sun　　　　　　Advisor：Dr. Wuu Yang

Department of Computer Science and Engineering
National Chiao Tung University

ABSTRACT

Asymmetric multicore systems had been studied as a new hardware platform toward performance power efficiency for the execution of application programs. Each core in the system has distinct performance and power characteristics. When exploiting asymmetric multicore systems, a major issue is to distribute threads to various cores. In this work, we build a pseudo asymmetric system by dynamic voltage frequency scaling (DVFS) mechanism on Intel core i7 920 for physical power measurement and implement a tool agent for regular JVM to form an asymmetric-aware JVM that supervises the execution of Java threads and migrates threads with fuzzy-control scheduler. For result inspection, we consider energy delay product (EDP) as a metric to reveal the compromise between performance and energy use. The major results include: Our fuzzy-control scheduler results in EDP benefit for scimark.fft.large and lower overall energy consumption.

誌　　　謝

能夠完成這篇論文，首先要感謝我的指導教授沒有忘記我的論文審定書放在哪裡。老師也花了很多心思在指導我，「以前有一個學生…」的故事還有在聽到我把機器弄壞以後的那句「那你完蛋了…」，老師的智慧與幽默，我永遠不會忘記的。老師人很好，在研究上也給我很多的資源、很大的自由和睿智的指導，能夠跟老師做研究是我的福氣。

在完成論文的過程，有許多人和許多的事物值得感謝，他們完整我這過程的精采時光。

給柏瞱學長與裕生學長，實驗室的智庫，總是能夠在適當的時候給我很適合的意見，沒有你們研究不會這麼順利。

給與我相處過的實驗室伙伴們，我很懷念那些每晚開戰的日子。實驗室的出遊和吃不完的宴席，這些在學的時間因為有你們更顯得精采。

給 221，與我同住一段時間的夥伴們，我很懷念那些開戰的夜晚。義氣、搥胸、rush，跟你們在一起永遠都不會沒勁。當然，那些學術交流也不能忘！

給同住在光復中學的家人們，那些步調悠閒的日子。劈西瓜、戰神三、阿布機爭奪戰和最討厭的胖胖。這才是生活！

給研究生活後期的朋友們，後期的苦悶，因為有你們才得以釋懷。我會懷念那些上系桌週報封面的照片。

我最深的感謝要給我的母親，有她在背後無悔的支持，我才能完成這一個人生的階段。

The work reported in this paper is partially supported by National Science Council, Taiwan, Republic of China, under grants NSC 99-2220-E-009-048- and NSC 99-2220-E-009-049-.

Contents

摘要 **ii**

Abstract **iii**

誌謝 **iv**

List of Figures **vii**

List of Tables **ix**

1 Introduction **1**

1.1 Asymmetric Multicore Systems 1

1.2 Toward Better Performance Power Efficiency 3

2 Related work **6**

3 Implementation **9**

3.1 Linux Performance Counter System and Java Threads 9

3.2 JVM Tool Interface 11

3.3 Migration Mechanisms 12

3.3.1 IPC scheduler 13

3.3.2 LLC-miss scheduler 16

4 Experiment **18**

4.1 Hardware Configuration 18

4.2 Software Environment 19

4.3 12V2 Rails Measurement 21

4.4 Benchmark Configuration 21

4.5 Destination Core Preference 22

5 Experiment Result **24**

5.1 IPC scheduler 25

5.1.1 Time increase 25

5.1.2 Energy Reduction 27

5.1.3 Energy Delay Product 30

5.2 LLC-miss scheduler 33

5.2.1 Time increase 33

5.2.2 Energy Reduction 33

5.2.3 Energy Delay Product 34

5.3 Migration overhead 34

6 Conclusion and Future works **37**

Bibliography **39**

List of Figures

1.1 An asymmetric multicore system that consists of four cores in a chip. 2

1.2 Data structure in the *agent* thread 3

1.3 Migration decision through fuzzy control system 4

3.1 Mapping between Java threads and Linux threads 10

3.2 JVM and the agent 11

3.3 Membership functions for IPC 13

3.4 Membership functions for IPC difference 13

3.5 Membership functions for destination core 14

3.6 Inference rules for IPC scheduler 15

3.7 LLC-miss scheduler modeling. The unit in LLC-miss domain and difference domain is 100 thousand. 17

4.1 Power measurement through 12V2 rails 20

4.2 Output variable for Asym A and Asym B 22

5.1 Time increase ratio under asym A for IPC scheduler 26

5.2 Core selection for IPC scheduler under asym A 26

5.3 Time increase ratio for IPC scheduler under asym B 28

5.4 Core selection for IPC scheduler under asym B 28

5.5 Energy reduction rate for IPC scheduler under asym A 29

5.6 Energy reduction rate for IPC scheduler under asym B 29

5.7 Relative energy delay product for asymmetric configuration A 30

5.8 Relative energy delay product for asymmetric configuration B 31

5.9 Time increase ratio for LLC-miss scheduler under asym A . . . 32

5.10 Core selection for LLC-miss scheduler under asym A 32

5.11 Energy reduction rate for LLC-miss scheduler under asym A . 33

5.12 Relative energy delay product for LLC-miss scheduler under asym A 34

5.13 Time increase in symmetric configuration with fuzzy-control scheduler 35

List of Tables

4.1 Intel i7 specification 19

4.2 4 configurations in our experiment 20

5.1 Base running time and power consumption under sym2.66 . . 25

5.2 The number of migrations per second 35

Chapter 1

Introduction

Asymmetric multicore systems lead to a better utilization of resource under proper use. The question is how to distribue these distinct cores for different requirement. This thesis presents a plug-in scheduler for regular JVM to form an asymmetry-aware JVM. This asymmetry-aware JVM utilizes underlying hardware performance counters for scheduling decisions. The goal is toward better energy delay product.

1.1 Asymmetric Multicore Systems

Computer processor development in the form of single-core systems had reached the limit in terms of power consumption, memory gap, and thermal control. Chip validation also becomes more and more difficult when IC chips get more and more complex. Although multicore systems had then become the solution for single core development difficulties, a problem concerning

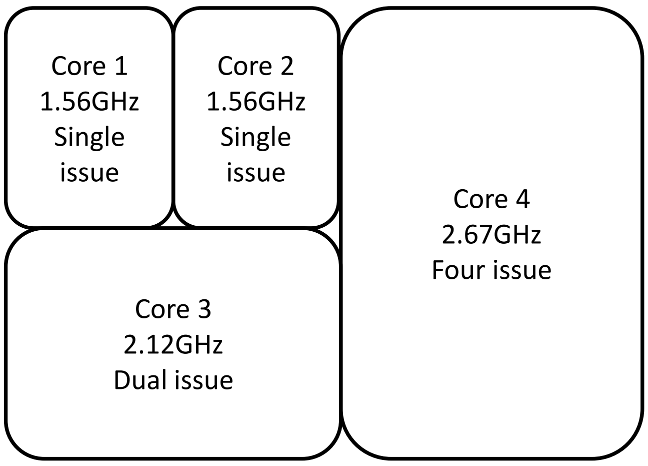


Figure 1.1: An asymmetric multicore system that consists of four cores in a chip.

the utilization of power and performance in multicore systems is raised since under-utilized cores waste energy. To address the energy-saving problem, many researchers proposed the asymmetric multicore architecture in order to achieve better performance power efficiency [7, 9, 11, 12, 14, 15].

Asymmetric multicore systems (ASYMS) consist of several cores (with identical or different ISA). These cores deliver different performance and power characteristics. Figure 1.1 shows a 4-core system. The larger core is usually faster and more complex and consumes more energy. Different cores fit different performance and/or energy requirements for different applications.

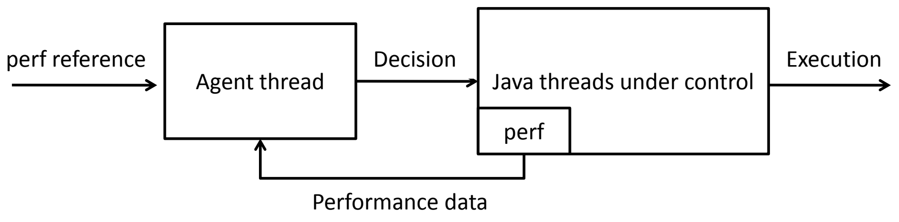


Figure 1.2: Data structure in the *agent* thread

1.2 Toward Better Performance Power Efficiency

When it comes to ASYMS, the software or the OS must be aware of this hardware asymmetry [6,7,13,20]. A more aggressive thread-scheduling policy is required compared to traditional scheduling policies whose sole purpose is load balancing among the cores. There are two issues in scheduling ASYMS: performance [6, 8, 10, 14, 18] and power efficiency [7, 11, 13, 15]. In this work, we propose a new scheduling mechanism based on fuzzy control for ASYMS. Our objective is to achieve better performance and power efficiency. We build a pseudo asymmetric single-ISA multicore system with Intel core-i7. Each core runs in different frequencies, controlled by the dynamic voltage frequency scaling (DVFS) mechanism [17] for physical power measurement.

We implement our schedulers through JVM tool interface for regular JVM to form an asymmetry-aware JVM. The asymmetry-aware JVM creates a monitor thread, called the *agent*, which monitors all active Java threads. Figure 1.2 gives the scenario of monitoring. Each java thread is bound with

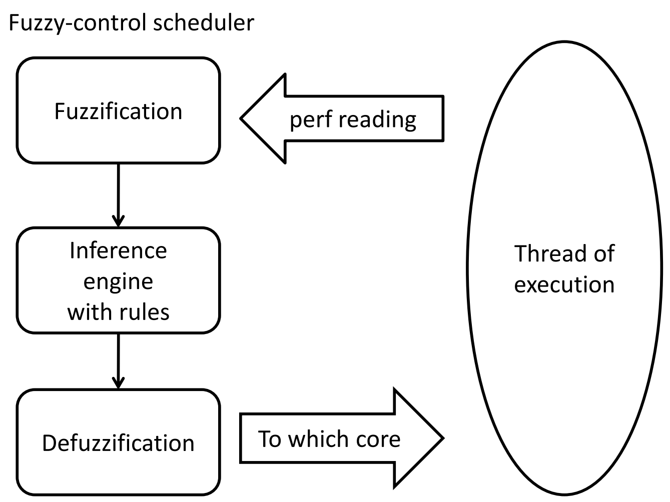


Figure 1.3: Migration decision through fuzzy control system

performance counters. Inside the agent thread is a fuzzy-control scheduler, which makes scheduling decisions for all Java threads based on statistics obtained from the hardware performance counter in Intel i7-920. Figure 1.3 shows the flow of the fuzzy-control scheduler. The fuzzy control scheduler makes the migration decision based on the imprecise statement: a thread should migrate to a more powerful core when the utilization of the current core is good. Here, we use the number of executed instructions per cycle (IPC) as the utilization index, although our early result shows that IPC may not be enough as an utilization index. Our heuristic is that threads with a high instruction-per-cycle index should be moved to faster cores [7]. A result in a previous study [7] also shows that these CPU-intensive programs will get more speedup when they run on cores with higher frequency.

We try to use the number of last-level cache (LLC) misses as another index too. The heuristic for this shceduler is to distribute java threads among slower cores when there are any amount of LLC misses.

The overhead caused by the agent is aslo under our consideration. One of the most significant overhead is the migration overhead since there might be a lot of thread migrations [14, 17].

This thesis is organized as follow: Chapter 2 describes important related works in asymmetric multicore systems era. Chapter 3 gives the overall implementation of asymmetry-aware JVM. The entire testing environment is depicted in Chapter 4. Chapter 5 shows the experiment results that include time increase ratio, energy use and EDP benefit. Chapter 6 gives the conclusion.

Chapter 2

Related work

An early study [11] addressed the potential of power reduction in single-ISA heterogeneous multicore architectures. During program execution, systems software dynamically chooses an appropriate core to meet the specific performance and power requirements [11]. Several off-the-shelf processor specifications were used to model the architectures for die space, performance, and power simulation. Our work starts with the same perspective of choosing appropriate cores for applications under performance requirements, but it differs in that we use a fuzzy-control scheduler in choosing cores.

A study of making use of the asymmetric multicore systems for power efficiency was in [7], in which the mechanisms of exploiting asymmetric multicore system were classified into thread-level parallelism specialization and microarchitecture specialization. In thread-level parallelism specialization, an application is divided into one or several sequential threads and several parallel threads. The sequential threads run on faster cores while the parallel threads run on slower cores. A performance study of this thread-level parallelism specialization was provided in [10]. In microarchitecture specialization, an application (especially for single-thread application) experiencing different phases [19] during execution would have its CPU-intensive phases run on faster cores or have its memory-intensive phases run on slower cores. Our work belongs to the category of microarchitecture specialization. We uses hardware performance counters [3,7], which were described to be an effective way for on-line utilization estimation, to determine the various phases during execution.

Dynamic Voltage Frequency Scaling (DVFS) is a sophisticated scheme toward power management. In contrast to traditional DVFS usage that changes the voltage and frequency dynamically depending on requirements, a different DVFS usage in [17] promises to provide better performance power efficiency. With a fixed but different voltage/frequency setup for each core in a multicore system, threads of the running program quickly migrate to different cores in order to adapt to the time-varying execution phases [17]. The work [17] is a form toward asymmetric sigle-ISA multicore systems.

Our work is similar to [17]. Since the asymmetric single-ISA multicore systems are not yet being built, a simple way to make the system asymmetric is to use different frequency setting for each core [7,9,18]. A difference is that we investigate the benefit of using the fuzzy heuristic to schedule the threads. The same DVFS usage to form a physical asymmetric system was used in several researches [6, 8, 13, 14].

The fuzzy-control theroy is a modern control method [22–24]. Previous work [16] revealed a fantastic way to combine the DVFS usage with the fuzzy control system at electronic level. In this work [16], the electricity current and the variation of the current are assumed to represent the work load of CPU and to serve as the input of the fuzzy control system. The supply voltage which is the output of the fuzzy control system changes dynamically on need. The use of variation gives an inspiration that we can use the instruction-per-cycle (IPC) and its history to predict future utilizaton and make mirgation decisions accordingly.

Chapter 3

Implementation

3.1 Linux Performance Counter System and Java Threads

Linux Performance Counter Subsystem is a kernel-based subsystem that provides a framework for collecting performance data. The performance events can be hardware performance counters or software events like page faults or context switches. Linux provides a simple system call `sys_perf_event_open` for these performance counters:

```
int sys_perf_event_open(
   struct perf_event_attr *hw_event,
   pid_t pid, int cpu, int group_fd,
   unsigned long flags);
```

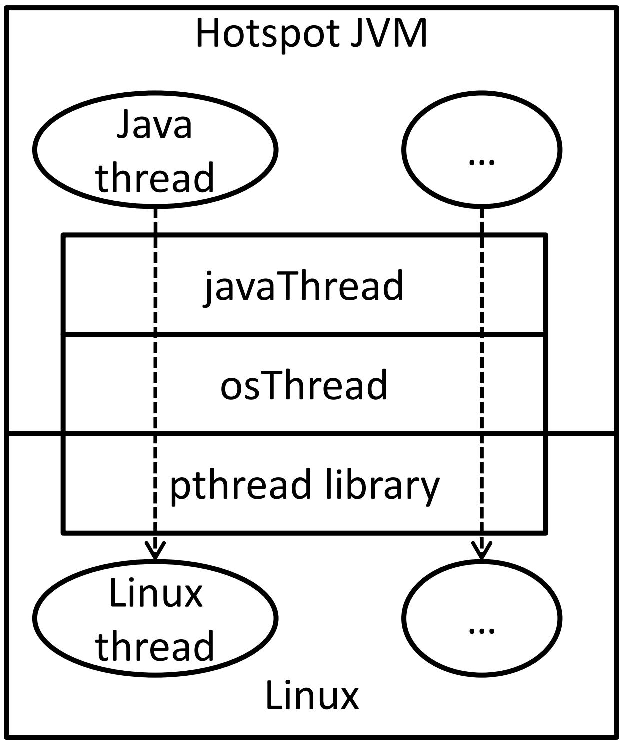


Figure 3.1: Mapping between Java threads and Linux threads

This system call returns a file descriptor through which a user program can read various hardware perfomance counters.

Both threads or processes can be monitored through the subsystem with the necessary privileges.

Since a Java process relies on Linux for process or thread creation in our environment, all the performance counters of spawned threads could be easily monitored.

Note that the mapping between Java threads and Linux threads is implementation-dependent. The JVM in our implementation is Java HotSpot Virtual Ma-

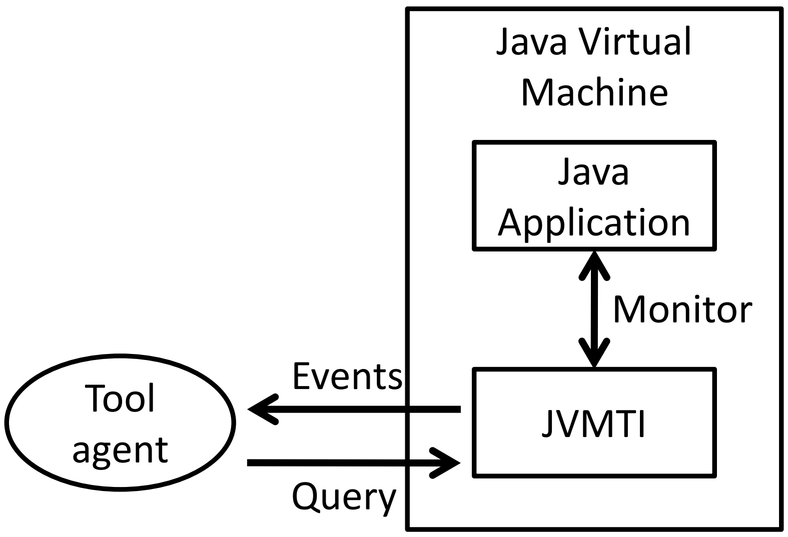


Figure 3.2: JVM and the agent

chine [4]. The mapping between Java threads and Linux threads is provided by the pthread library. It is an 1-1 mapping as shown in Figure 3.1. This means that we can use Linux Performance Counter Sytem to monitor the Java threads.

3.2 JVM Tool Interface

The JVM Tool Interface (JVMTI) provides a programming interface to inspect the state and to control the execution of applications running in the Java virtual machine. The monitor thread, called the *agent*, is a dynamic linking library, which can be loaded by any JVMTI-capable JVM as shown in Figure 3.2. Since the mapping between Java threads and Linux threads is 1-1, we may monitor the creation of Java threads through JVMTI and register the mapping between Java threads and Linux threads. This mapping is used to interpret the performance counter statistics.

3.3 Migration Mechanisms

We implement two fuzzy-control scheduler fors our asymmetric-aware JVM: The IPC scheduler and the LLC-miss scheduler. Each scheduler is a closed control loop that takes different input and makes migration decisions periodically. These fuzzy-control schedulers act like a headphone volume control that one will shift the volume left when the headphone sounds loud, or shift right otherwise.

A fuzzy-control scheduler consists of three parts: a fuzzification part, an inference engine with a set of rules, and a defuzzification part. The fuzzification part converts a quantitative value into a qualitative value. Several linguistic variables may then be used in the inference engine. The inference engine comes with a set of rules, which are based on human knowledge over the controlled object to infer output fuzzy sets. The defuzzification part converts the output sets to a crisp number, which is used to control the intended device.

Here, we use the IPC fuzzy-control scheduler as an explanation for heuris-

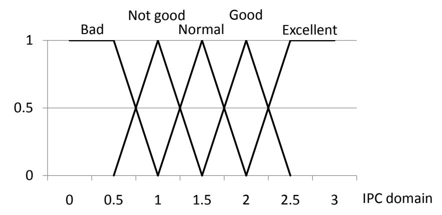


Figure 3.3: Membership functions for IPC

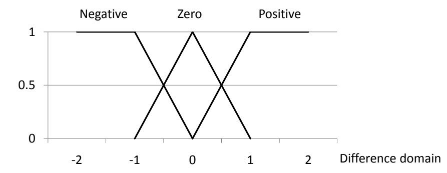


Figure 3.4: Membership functions for IPC difference

tic construction.

3.3.1 IPC scheduler

In the fuzzification part, there are *linguistic variables*. A linguistic variable stands for a human concept like “age” or “temperature” and has values like 23 years old or $31°C$. A linguistic variable is associated with one or more membership functions like “old” or “hot” that specify the degree a given input value describes the concept, for example, how old or how hot.

We have two input linguistic variables in our case: the number of instructions executed per cycle (IPC) and the IPC difference. Since each core in i7-920 processor is ideally four-issue capable, we define a general domain for IPC values within 0-3.5. The IPC variable divides the general domain

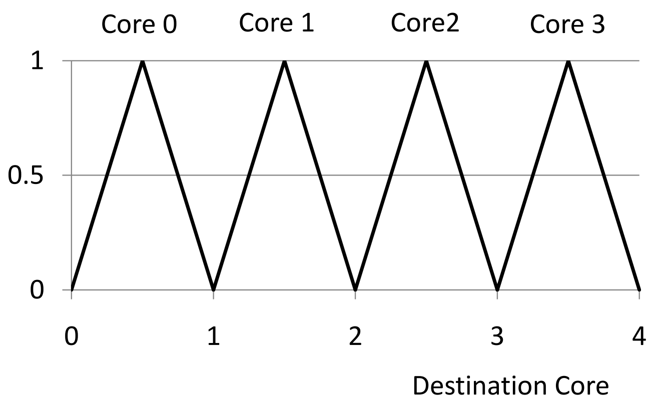


Figure 3.5: Membership functions for destination core

of IPC values into five membership functions. These are `bad`, `not good`, `normal`, `good` and `excellent`, as shown in Figure 3.3. Althought there are many possible choices of the shapes of membership functions, like Gaussian, trapezoid, triangular, etc., all these do not influence the output in a significant way according to [16]. So we choose the triangular and trapezoid membership functions in our case because of lower computational complexity of these shapes. The same choices are picked for the membership function for IPC difference, as shown in Figure 3.4.

The overlaps between these membership functions are 50%. This implies that an IPC input value will hit at most two membership functions. Similarly for the IPC difference. Thus, the inference engine will have four different rules at most to do an inference.

The design of output variable depends highly on the target ASYMS. Figure 3.5 gives the basic design of output variable for destination core in

RULES	Bad	Not good	Normal	Good	Excellent
Neg	0	0	1	2	3
Zero	0	1	2	2	3
Pos	0	2	3	3	3

Figure 3.6: Inference rules for IPC scheduler

our environment. Since we will have four cores with different frequencies in our experiment environment, the output variable, *destination core*, should have four membership functions: `core 0`, `core 1`, `core 2`, `core 3`. The number of membership functions for destination core depends on how many performance capabilities we can differentiate from the underlying ASYMS. In our case, we could have four different cores at most in terms of clock frequency. Note that we can define the four membership functions with different area to reflect the preference for cores. For example, if there is one core with much slower frequency than the three, we would not prefer to use the much slower one and thus define the associated membership function with smaller area to influence the decision made by the final defuzzification part.

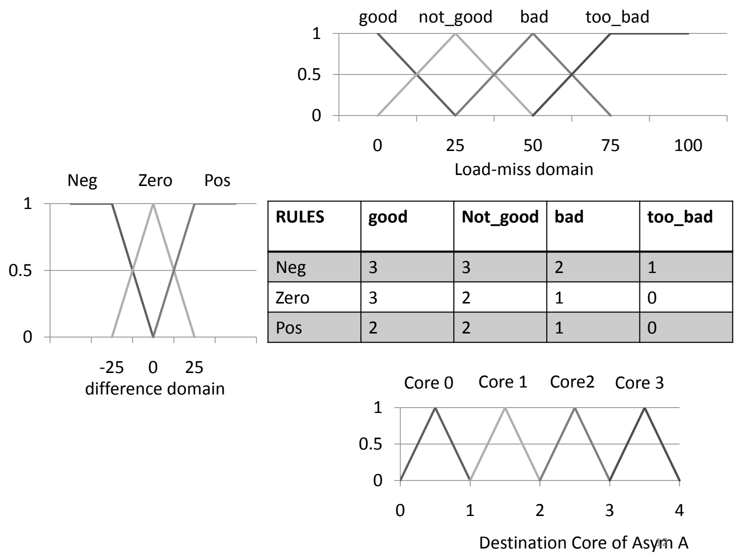
The inference enegine consists of inference rules, which are based on our domain knowledge, to infer implied membership functions. Since we have five membership functions in the IPC variable and three in the IPC variation variable, the inference engine should consist of fifteen rules. Figure 3.6

presents the 15 synthetic rules. For example, if the IPC is good and the difference is positive, the destination core is core 3. The inference logic used in our case is Zadeh inference engine [22]. Finially, the inference will usually produce four implied membership functions which will then be used in defuzzification process.

The defuzzification part converts the implied fuzzy sets into a crisp number that serves as a control signal. There are several common defuzzification methods, such as centroid, average center, mean of maxima, etc. [16, 22]. We choose the centroid method because it is popular and this method will react to different area of membership function when we have preference for some cores [22]. The calculated centroid will fall within one of the membership functions. Controled threads will be migrated to the destination accordingly. For instance, if the calculated centroid is 2.3, the thread is migrated to core 2.

3.3.2 LLC-miss scheduler

As for LLC-miss scheduler, we use the number of `llc-load-miss`, which is provided by the Linux performance counter system, as the input of fuzzy-control scheduler. (the system provides LLC events like load-reference, store-reference, load-miss and store-miss. We use the llc-load-miss as a starting point for control modeling.) Figure 3.7 gives the overall design of LLC-miss scheduler. The major problem in designing the LLC-miss scheduler is to answer the question: how many amount of LLC misses do we consider it a



RULES	good	Not_good	bad	too_bad
Neg	3	3	2	1
Zero	3	2	1	0
Pos	2	2	1	0

Figure 3.7: LLC-miss scheduler modeling. The unit in LLC-miss domain and difference domain is 100 thousand.

great amount. We choose the amount in Figure 3.7 from profiling data of benchmarks in our experiment. As a net effect, threads experiencing more than 3 hundred thousand LLC misses in one schedule period are scheduled to slower cores. The difference is derived from current and previous LLC misses as a prediction of increasing or decreasing trend. The design in output variable, inference part, and defuzzification part is the same in IPC scheduler.

Chapter 4

Experiment

4.1 Hardware Configuration

The hardware platform we use in our experiment is an Intel core i7-920 processor, which has four cores in a chip. A detail list of specification is shown in Tabele 4.1. Note that We turn off both the Intel Turbo Boost Technology and the Intel Hyper-Threading Technology to eliminate their effect on performance and power consumption.

It should be mentioned that there is a L3 shared cache in Intel i7-920. According to previous research [21], the shared cache prevents serious performance loss from thread migration. So we expect that we will not suffer much performance loss from thread migration in our experiments.

Table 4.1: Intel i7 specification

Processor Number	i7-920
# of cores	4
# of threads	8
Clock Speed	2.66 GHz
Max Turbo Frequency	2.93 GHz
L1 Cache	32KB L1 data, 32KB L1 instruction per core
L2 Cache	256KB per core, inclusive
L3 Cache	8 MB shared cache
Lithography	45 nm
Motherboard	GIGABYTE GA-EX58-UD3R

4.2 Software Environment

The operating system used in the experiments is Debian GNU/Linux built upon kernel version 2.6.32. The kernel provides the acpi-cpufreq module for controlling the frequency of cores. We use this module to set up the asymmetric environment. The available frequencies provided by this module for the hardware include 2.66GHz, 2.53GHz, 2.39GHz, 2.26GHz, 2.13GHz, 2.00GHz, 1.86GHz, 1.73GHz, and 1.60GHz. Table 4.2 shows the 4 configurations used in our experiments. The symmetric configuration sym2.66 consists of four cores, all running with the highest frequency. The asymmetric configuration A makes use of the highest four distinct frequencies. We also test our control system under a more extreme environment like asymmetric configuration B.

Table 4.2: 4 configurations in our experiment

	CPU 0	CPU 1	CPU 2	CPU 3
Sym2.66	2.66GHz	2.66GHz	2.66GHz	2.66GHz
Asym A	2.26GHz	2.39GHz	2.53GHz	2.66GHz
Sym2.26	2.26GHz	2.26GHz	2.26GHz	2.26GHz
Asym B	1.60GHz	2.13GHz	2.39GHz	2.66GHz

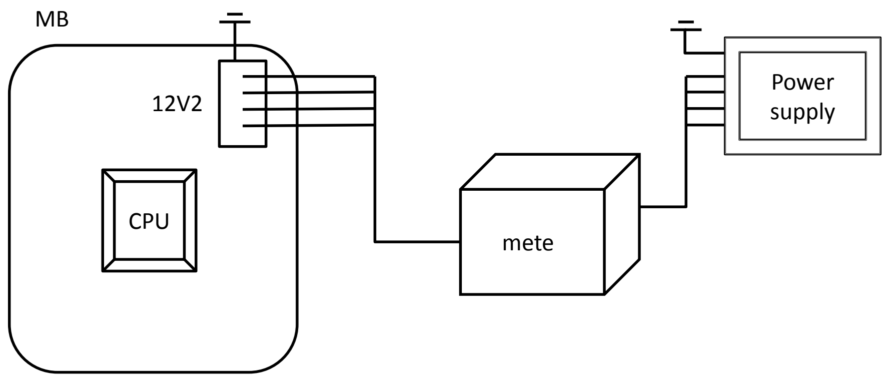


Figure 4.1: Power measurement through 12V2 rails

Configuration sym2.26 is provided to realize the worst performance loss in Asym A.

Note that we rely on the system call `sched_setaffinity` to do the actual migration. By specifing a CPU mask to target thread, we are allowed to bind target thread to one or a subset of all logical cores under proper privilege.

4.3 12V2 Rails Measurement

According to EPS12V specifications [1], i7 processor power is provided by 12V2 rails. We make an instrument to these rails as shown in Figure 4.1. The power meter used in our experiments is NI-4065 digital multimeter. The meter provides both current and timing data. The power consumption is derived through these data. Project SIKULI [2] is used in the measurement for test automation.

4.4 Benchmark Configuration

SPECjvm2008 is our experimental benchmark. We configured the benchmark suite to use lagom [5] workload, in which every application in the suite does a fixed amount of work in one run. The number of threads in each application is set to one so that there should be one active computational thread during application execution. An exception is *sunflow* because multiple active threads is required during the execution. Note that we did not provide the measurement data of *sunflow* since all the cores in our hardware require running when testing this application, and the current during the measurement is beyond the capability of our power meter.

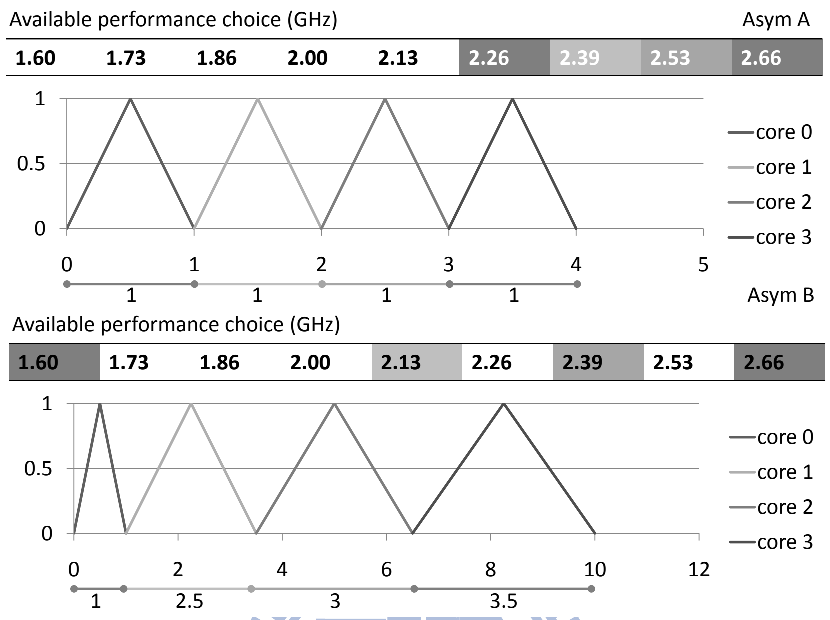


Figure 4.2: Output variable for Asym A and Asym B

4.5 Destination Core Preference

As stated before, we desine the output variable, destination core, to react to the different perfomance gap between cores. When there is a big performance gap between two destination core, a reasonable selection of cores is trying not to select the core which is much slower. As a result, we design two output variables for Asym A and Asym B as shown in Figure 4.2. As for Asym B, each destination membership function has different bottom which results in different area. The bottoms are calculated as follows: The bottom of slowest destination core is always one. If there is one more choice between two destination cores, we increase the bottom of faster one by 0.5. For example, there are three more choices between core 0 and core 1 in asym B, so the bottom of core 1 is calculated as $1 + 1.5 = 2.5$, and so on. In this way, the core 0 will have less chance to be selected.

Chapter 5

Experiment Result

Table 5.1 gives the baseline data (execution time and power consumption), which is measured in the symmetric configuration sym2.66. Although we configure the SPECjvm2008 to use only one benchmark thread, each application may still create multiple threads. These threads stay in the wait state most of the time. Exceptional applications *compiler.compiler* and *compiler.sunflow* consume more power than others during one run because the two applications require two cores to run two threads simultaneously in some time.

Note that we test the LLC-miss scheduler in asym A for now. The experiment results for IPC scheduler in both asym A and asym B are provided.

Table 5.1: Base running time and power consumption under sym2.66

application	sec	watt	application	sec	watt
cmpiler.compiler	18.04	22.5	scimark.fft.large	28.73	10.9
compiler.sunflow	32.08	17.9	scimark.lu.large	33.6	14.3
compress	78.15	13.4	scimark.sor.large	97.42	11.1
crypto.aes	94.96	14.1	scimark.sparse.large	50.44	13.7
crypto.rsa	148.6	14.6	scimark.monte_carlo	1609	15.2
crypto.signverify	116.8	13.7	scimark.fft.small	86.29	14.4
derby	96.49	11.9	scimark.lu.small	99.65	16.3
mpegaudio	128.6	13.6	scimark.sor.small	112.2	10.9
serial	46.4	13.9	scimark.sparse.small	48.86	15.9
xml.transform	28.70	17	xml.validation	29.35	14.9

5.1 IPC scheduler

5.1.1 Time increase

Figure 5.1 gives the ratio of the execution time in asymmetric configuration A relative to the baseline (i.e., the symmetric configuration sym2.66) under our fuzzy-control scheduler. We also provide the execution time in symmetric configuration sym2.26. (Note that 2.26GHz is the lowest frequency in configuration asym A.) This gives an upper bound in time increase ratio. Smaller increase in execution time should indicate that memory operations are more critical for the application. Figure 5.2 shows the breakdown of the

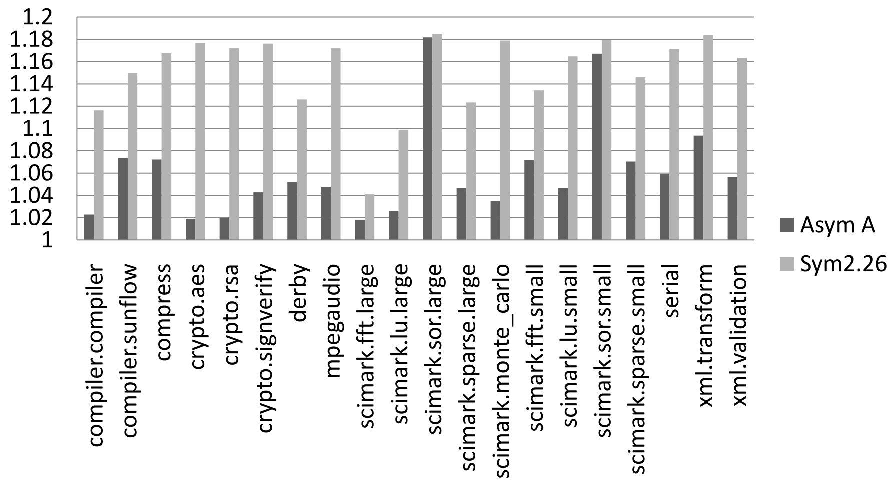


Figure 5.1: Time increase ratio under asym A for IPC scheduler

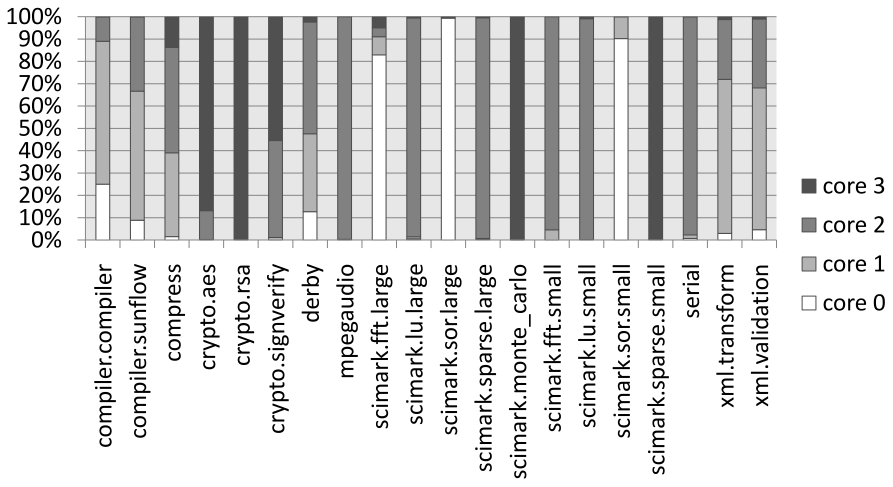


Figure 5.2: Core selection for IPC scheduler under asym A

execution time for each application that are dedicated to each core with the fuzzy-control scheduler under asym A. This demonstrates the selection made by the fuzzy-control scheduler.

The time increase ratio in Sym2.26 indicates that application *scimark.fft.large* and *scimark.lu.large* are less sensitive to the change of frequency, especially for *scimark.fft.large*. The fuzzy-control scheduler made a right choice to use slowest core mostly for it.

As for *scimark.sor.large* and *scimark.sor.small*, the scheduler choose slower cores since these applications remain in low IPC state during the whole execution. In contrast, application *crypto.aes*, *crypto.rsa*, *scimark.monte_carlo* and *scimark.sparse.small* are forced to use fastest core.

For others applications, the choices made by the fuzzy-control scheduler result in less performance loss than that in sym2.26 because the use of each core is in a balance.

Figure 5.3 shows the time increase ratio under asymmetric configuration B. The breakdown of execution time in each core for asym B is provided in Figure 5.4. The results are similar to that under asym A except that core 0 has less chance to be selected. This prevents a great performance loss from choosing the slowest core like *scimark.sor.** did under configuration asym A.

5.1.2 Energy Reduction

We calculate the average energy consumption from the power meter readings. Figure 5.5 gives the energy reduction rate in asymmetric configuration A

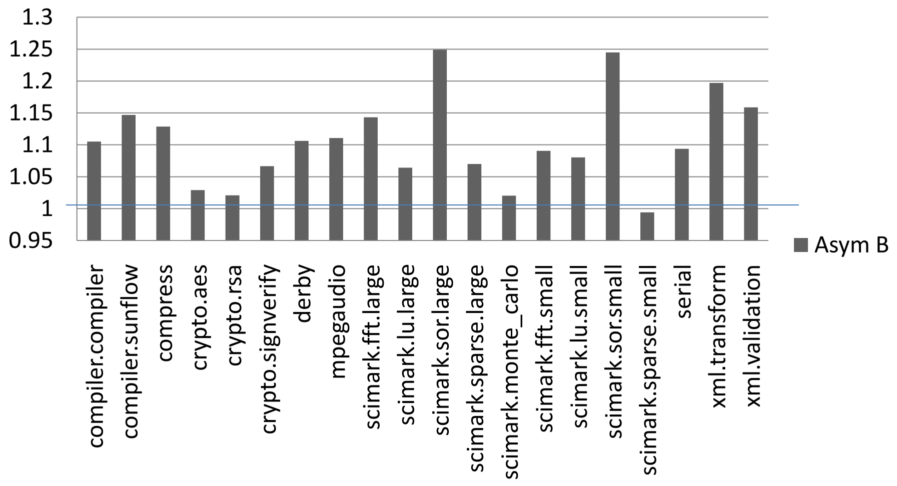


Figure 5.3: Time increase ratio for IPC scheduler under asym B

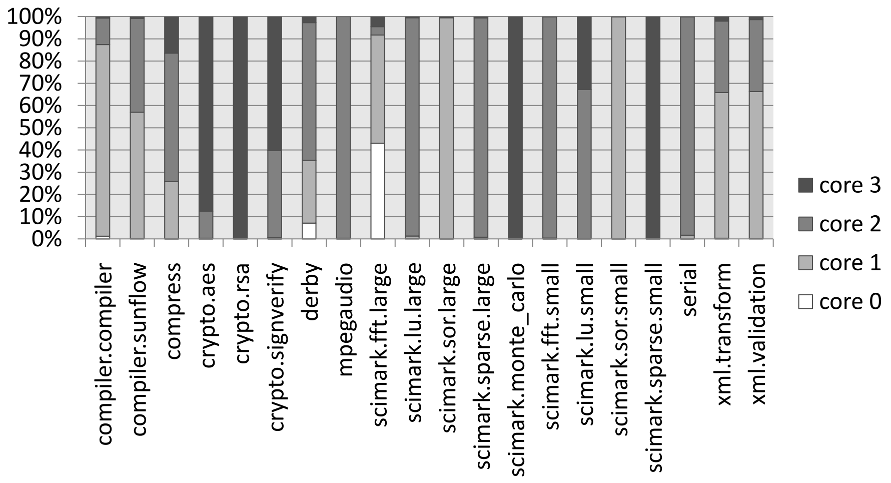


Figure 5.4: Core selection for IPC scheduler under asym B

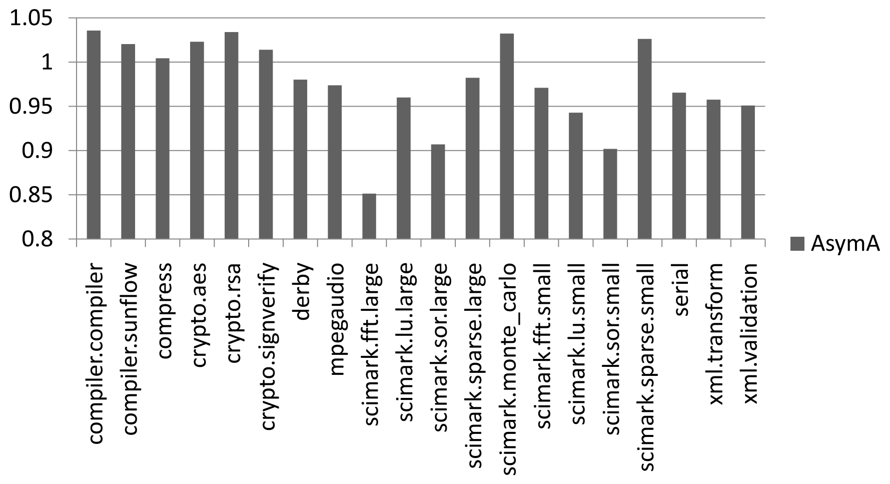


Figure 5.5: Energy reduction rate for IPC scheduler under asym A

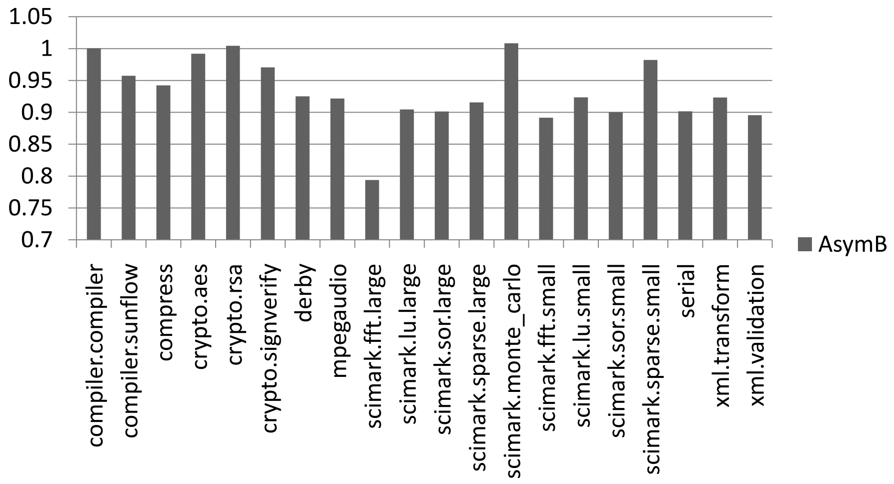


Figure 5.6: Energy reduction rate for IPC scheduler under asym B

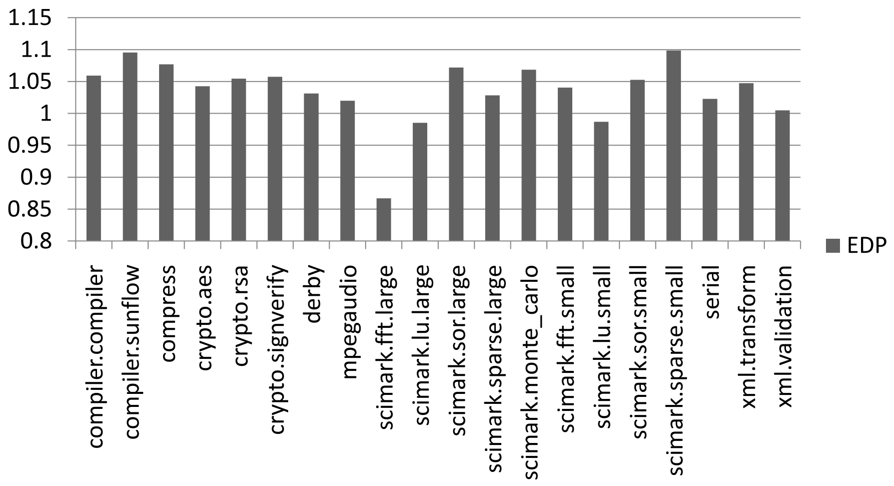


Figure 5.7: Relative energy delay product for asymmetric configuration A

relative to the baseline. The energy reduction result in asym B is given in Figure 5.6. Energy consumption is reduced to 97% and 93% at average in asym A and asym B respectively.

5.1.3 Energy Delay Product

We compare EDP in asymmetric configuration A under the fuzzy-control scheduler and EDP in the symmetric configuration sym2.66 without a scheduler. Figure 5.7 presents the result in EDP benefit. The fuzzy-control scheduler chooses faster cores mostly for *crypto.**, *mpegaudio*, *scimark.monte_carlo*, *scimark.lu.small* and *scimark.sparse.small* since these applications exhibit high IPC state during entire execution. Although the scheduler makes right choices for these applications, there is a slight increase in EDP. Application *scimark.fft.large* reveals a positive benefit in EDP reduction.

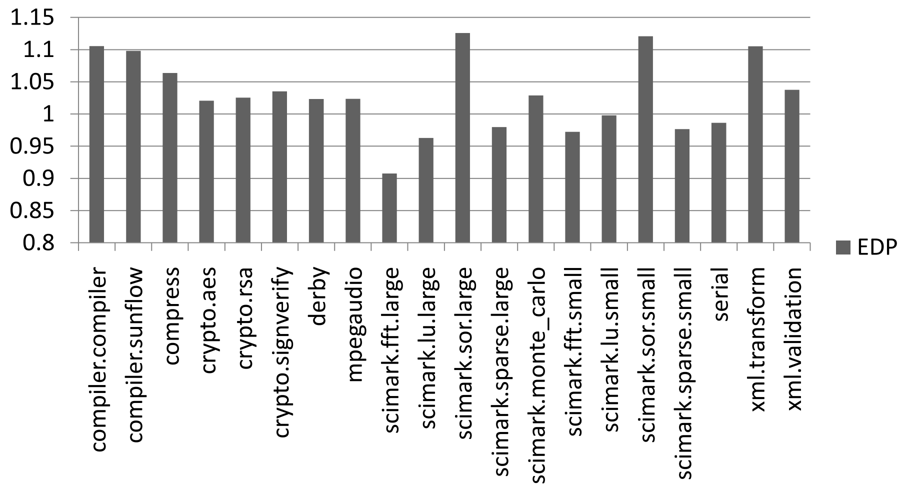


Figure 5.8: Relative energy delay product for asymmetric configuration B

The EDP reduction results for other applications in asym A are not well, especially for scimark.sor.* which is known as CPU-intensive application in Figure 5.1. A post-review on performance counter statistic of these applications shows the IPC may not always indicate the right intensiveness (CPU or memory). This should lead to a new design of fuzzy scheduler that takes other metric (e.g., last-level cache miss) into consideration.

The EDP reduction rate in asym B is provied in Figure 5.8. There are more applications which show positive benefit than that in asym A. For these applications, the use of lower frequency might increase the EDP benefit in their memory-intensive part.

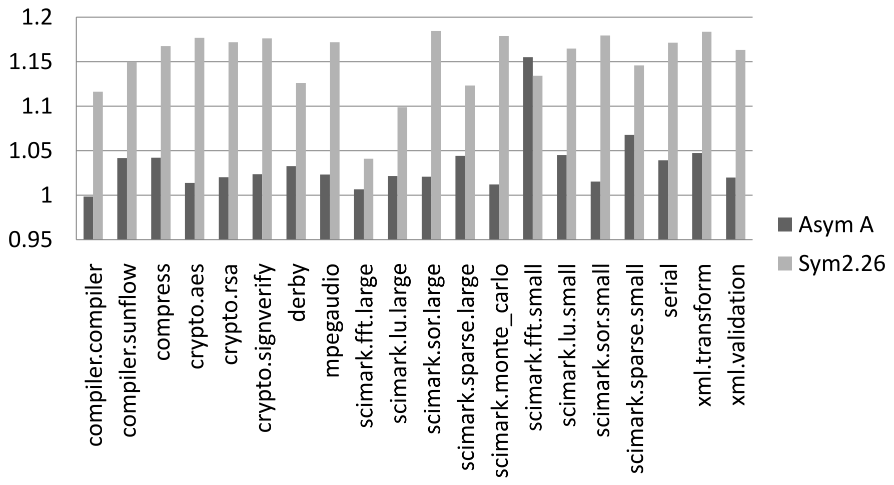


Figure 5.9: Time increase ratio for LLC-miss scheduler under asym A

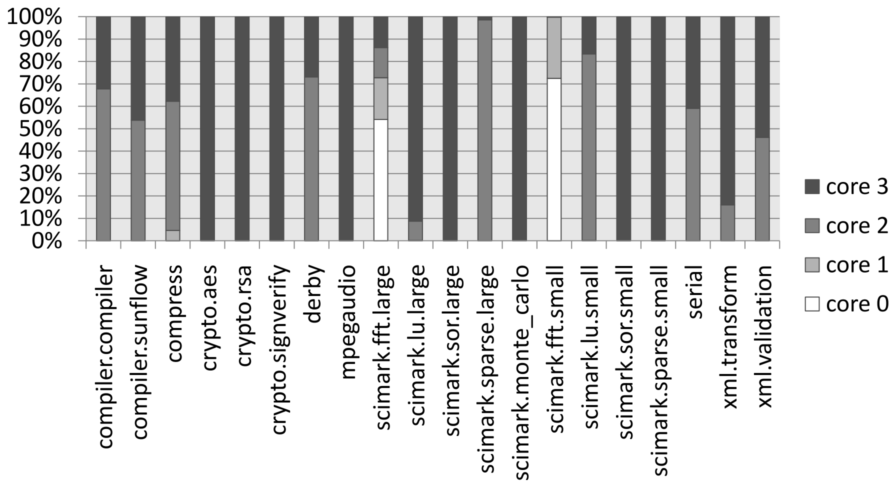


Figure 5.10: Core selection for LLC-miss scheduler under asym A

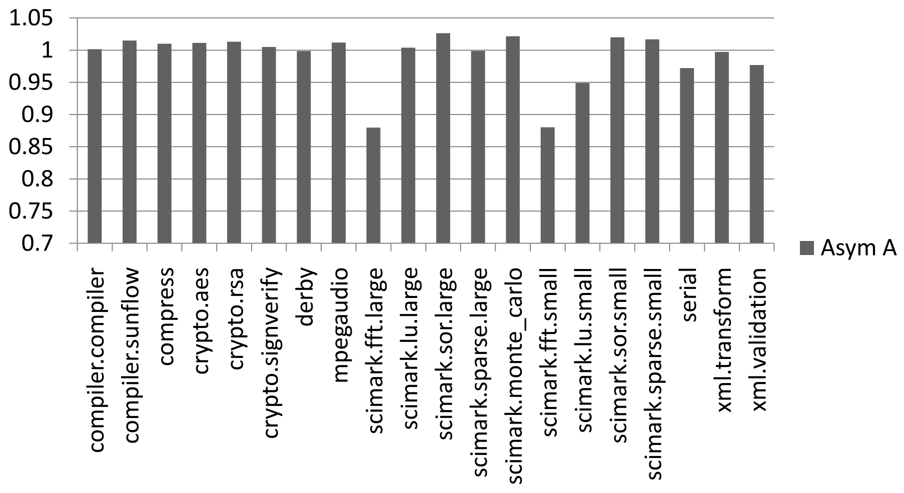


Figure 5.11: Energy reduction rate for LLC-miss scheduler under asym A

5.2 LLC-miss scheduler

5.2.1 Time increase

Figure 5.9 and Figure 5.10 give time increase ratio and core selection for LLC-miss scheduler under asym A respectively. Considering the intensiveness shown in sym2.26, the scheduler makes right decisions for most of the applications except *scimark.fft.small*.

5.2.2 Energy Reduction

Since the scheduler decides to use faster core for most of the applications, there are little benefits in energy reduction.

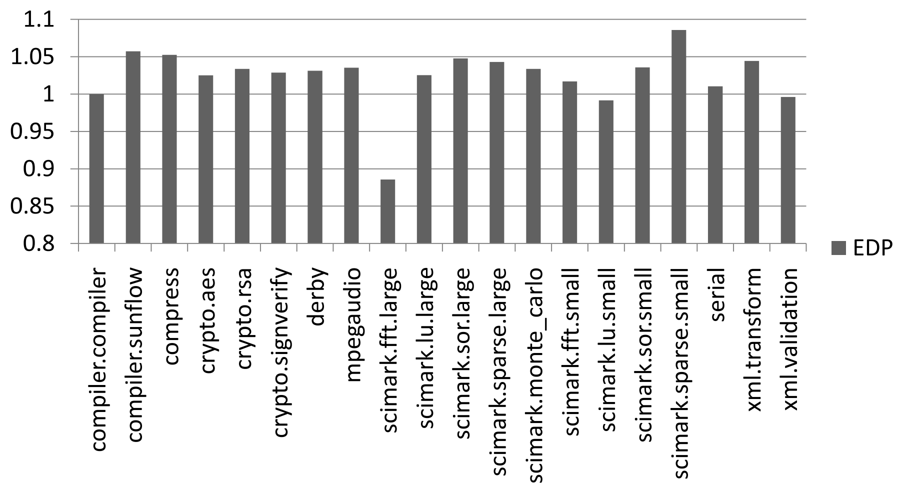


Figure 5.12: Relative energy delay product for LLC-miss scheduler under asym A

5.2.3 Energy Delay Product

Figure 5.12 presents the EDP benefit for LLC-miss scheduler. Again, we have best benefit for *scimark.fft.large*. Strictly speaking, *scimark.fft.large* is the only memory-intensive application in SPECjvm2008. Others are that sensitive to the change of frequency. This leads to a question that how do we identify this kind of memory-intensive with performance counter statistic, especially in a short time period.

5.3 Migration overhead

We measure the overhead by running the benchmarks in the symmetric configuration sym2.66 with the IPC scheduler since this scheduler made more

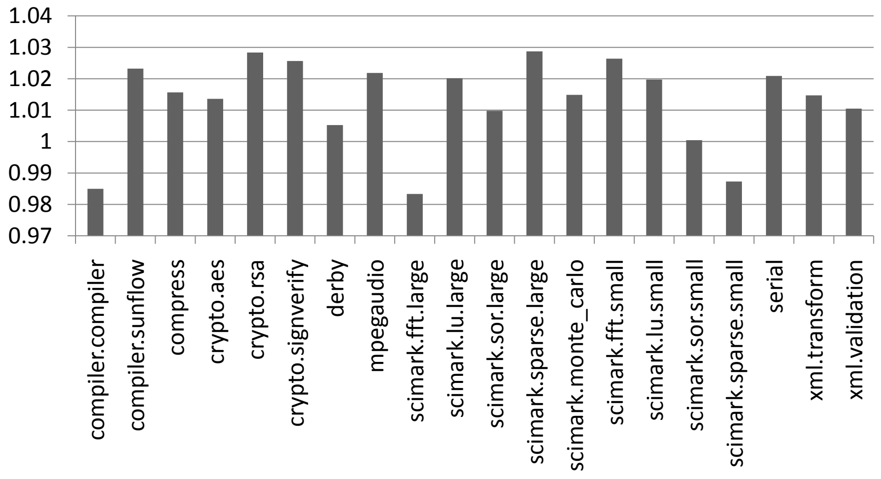


Figure 5.13: Time increase in symmetric configuration with fuzzy-control scheduler

Table 5.2: The number of migrations per second

application	migr/s	application	migr/s
cmpiler.compiler	64.26	scimark.fft.large	32.94
compiler.sunflow	69.12	scimark.lu.large	82.38
compress	65.92	scimark.sor.large	78.54
crypto.aes	11.61	scimark.sparse.large	85.86
crypto.rsa	0.27	scimark.monte_carlo	0.16
crypto.signverify	46.43	scimark.fft.small	78.70
derby	65.40	scimark.lu.small	89.82
mpegaudio	86.06	scimark.sor.small	69.34
serial	82.95	scimark.sparse.small	0.30
xml.transform	62.48	xml.validation	59.40

migrations than LLC-miss scheduler. Figure 5.13 gives the increase ratio of execution time and Table 5.2 shows the number of migrations per sec. Although the IPC scheduler causes massive migrations every second for most of the applications, we do not see relevent increase in execution time. This result should be consistent with previous research [21] which stated the advantage of having a shared cache between cores for thread migration overhead prevention. We believe that the increase of execution time in the asymmetric configuration should be due to the use of lower frequency.

Chapter 6

Conclusion and Future works

Asymmetric multicore systems had been studied as a new hardware platform toward performance power efficiency. The major question is how to deal with the scheduling problem [7,8,14,18,20]. As a result, we present a fuzzy-control scheduler based on performance counter statistic on JVM. The experiment results shows that the fuzzy-control schedulers exhibit EDP benefit for one memory-intensive application. One of the major future work is to precisely identify this kind of memory-intensive application with performance counter statistic.

Although we do not provide power reduction rate data, we achieve the best 85% power reduction rate at average for entire SPECjvm2008. This is important to the development of ASYMS. Although the fuzzy-control schedulers make right decisions for several applications, there is still improvement we can make in th future work. The results also show that the migration overhead is negligible in our ASYMS and the major cause of performance degradation is the use of lower frequency.

Migration overhead should be a factor in designing the membership function. However, our environment consists of several cores in the same chip. These cores share some cache, such as L3 cache. Thus, the migration overhead is expected to be quite insignificant. When the fuzzy-control scheduler is scaled up to thousands of chips and thousands of cores, the membership functions should be adjusted to take into account the migration overhead.

Since we are exploiting the micro-architecture specialization, a future work could contain experiments on exploiting the thread-level parallelism specialization. Future works should also include more detailed tests in different frequencies to understand the effect on performance and fine tuning on our fuzzy-control scheduler.

The work here is incomplete. One should first discover the relationship between performance counter statistic and program intensiveness. A more complete work may be to include the learning process of fuzzy control to be more certain about the relationship above.

Bibliography

[1] Entry-level power supply specification, [online]. Available: http://en.wikipedia.org/wiki/Entry-Level_Power_Supply_Specification.

[2] Project sikuli, [online]. Available: http://sikuli.org/.

[3] Performance counters for linux, [online]. Available: https://perf.wiki.kernel.org/index.php/Main_Page.

[4] Java hotspot virtual machine, [online]. Available: http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136373.html.

[5] Specjvm2008 user's guide, [online]. Available: http://www.spec.org/jvm2008/docs/UserGuide.html.

[6] Saisanthosh Balakrishnan, Ravi Rajwar, Mike Upton, and Konrad Lai. The impact of performance asymmetry in emerging multicore architectures. *SIGARCH Comput. Archit. News*, 33(2):506–517, 2005.

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

[8] Soraya Ghiasi, Tom Keller, and Freeman Rawson. Scheduling for heterogeneous processors in server systems. In *CF '05: Proceedings of the 2nd ACM International Conference on Computing Frontiers*, pages 199–210, 2005.

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

[10] Mark D. Hill and Michael R. Marty. Amdahl's law in the multicore era. *Computer*, 41(7):33–38, 2008.

[11] R. Kumar, K.I. Farkas, N.P. Jouppi, P. Ranganathan, and D.M. Tullsen. Single-ISA heterogeneous multi-core architectures: the potential for processor power reduction. In *MICRO-36 '03: Proceedings of 36th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 81 – 92, Dec 2003.

[12] Rakesh Kumar, Dean M. Tullsen, Norman P. Jouppi, and Parthasarathy Ranganathan. Heterogeneous chip multiprocessors. *Computer*, 38:32–38, 2005.

[13] Viren Kumar and Alexandra Fedorova. Towards better performance per watt in virtual environments on asymmetric single-ISA multi-core systems. *SIGOPS Oper. Syst. Rev.*, 43(3):105–109, 2009.

[14] Tong Li, Dan Baumberger, David A. Koufaty, and Scott Hahn. Efficient operating system scheduling for performance-asymmetric multi-core architectures. In *SC '07: Proceedings of the 2007 ACM/IEEE Conference on Supercomputing*, pages 1–11, 2007.

[15] Jeffrey C. Mogul, Jayaram Mudigonda, Nathan Binkert, Parthasarathy Ranganathan, and Vanish Talwar. Using asymmetric single-ISA CMPs to save energy on operating systems. *IEEE Micro*, 28(3):26–41, 2008.

[16] H.R. Pourshaghaghi and J.P. de Gyvez. Dynamic voltage scaling based on supply current tracking using fuzzy logic controller. In *ICECS '09: Proceedings of 16th IEEE International Conference on Electronics, Circuits, and Systems*, pages 779 –782, 2009.

[17] Krishna K. Rangan, Gu-Yeon Wei, and David Brooks. Thread motion: fine-grained power management for multi-core systems. *SIGARCH Comput. Archit. News*, 37(3):302–313, 2009.

[18] 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(2):66–75, 2009.

[19] Timothy Sherwood, Erez Perelman, Greg Hamerly, Suleyman Sair, and Brad Calder. Discovering and exploiting program phases. *IEEE Micro*, 23(6):84–93, 2003.

[20] Richard Strong, Jayaram Mudigonda, Jeffrey C. Mogul, Nathan Binkert, and Dean Tullsen. Fast switching of threads between cores. *SIGOPS Oper. Syst. Rev.*, 43:35–45, April 2009.

[21] 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 *ISPASS '09: Proceedings of IEEE International Symposium on Performance Analysis of Systems and Software*, pages 123 –132, 26-28 2009.

[22] Li-Xin Wang. *A course in fuzzy systems and control*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1997.

[23] Lotfi A. Zadeh. Fuzzy sets and their applications to cognitive and decision processes. *Academic Press*, pages 1–39, 1975.

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