2013

Exploiting heterogeneity in Chip-Multiprocessor Design

Ying Zhang
Louisiana State University and Agricultural and Mechanical College, yzhan29@tigers.lsu.edu

EXPLOITING HETEROGENEITY IN CHIP-MULTIPROCESSOR DESIGN

A Dissertation

Submitted to the Graduate Faculty of the
Louisiana State University and
Agricultural and Mechanical College
in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

in

The Division of Electrical and Computer Engineering
School of Electrical Engineering and Computer Science

by
Ying Zhang
B.E., Huazhong University of Science and Technology, Wuhan, China 2006
M.E., Huazhong University of Science and Technology, Wuhan, China 2008
December 2013

ACKNOWLEDGEMENTS

I would like to dedicate this dissertation to my parents, my wife, and my parents-in-law, for their continuous support and encouragement throughout my entire life.

This dissertation could not have been completed without the help and support from a lot of people that I am grateful to. First of all, I would like to thank my advisor, Dr. Lu Peng, for his guidance and suggestions during my Ph.D. study. All of the works presented in this dissertation came from constant support and discussions with Dr. Peng. I would also like to thank Dr. Bin Li from Department of Experimental Statistics for all the collaboration we had and Dr. Xin Fu from University of Kansas for her invaluable comments on this research. Furthermore, I want to thank Dr. Jerry Trahan, Dr. Ramachandran Vaidyanathan, and Dr. Ronald F. Malone (the professors in my committee) for spending time supervising my dissertation and attending my defense.

I am thankful to the Department of Electrical and Computer Engineering for providing assistantship throughout my Ph.D. study.

Finally, I would like thank all the friends I met at LSU for making my life here wonderful and memorable.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

In the past decade, semiconductor manufacturers are persistent in building faster and smaller transistors in order to boost the processor performance as projected by Moore's Law. Recently, as we enter the deep submicron regime, continuing the same processor development pace becomes an increasingly difficult issue due to constraints on power, temperature, and the scalability of transistors. To overcome these challenges, researchers propose several innovations at both architecture and device levels that are able to partially solve the problems. These diversities in processor architecture and manufacturing materials provide solutions to continuing Moore's Law by effectively exploiting the heterogeneity, however, they also introduce a set of unprecedented challenges that have been rarely addressed in prior works.

In this dissertation, we present a series of in-depth studies to comprehensively investigate the design and optimization of future multi-core and many-core platforms through exploiting heterogeneities. First, we explore a large design space of heterogeneous chip multi-processors by exploiting the architectural- and device-level heterogeneities, aiming to identify the optimal design patterns leading to attractive energy- and cost-efficiencies in the pre-silicon stage. After this high-level study, we pay specific attention to the architectural asymmetry, aiming at developing a heterogeneity-aware task scheduler to optimize the energy-efficiency on a given single-ISA heterogeneous multi-processor. An advanced statistical tool is employed to facilitate the algorithm development. In the third study, we shift our concentration to the device-level heterogeneity and propose to effectively leverage the advantages provided by different materials to solve the increasingly important reliability issue for future processors.

CHAPTER 1. INTRODUCTION

Moore's Law, which describes that the number of transistors integrated on chip will be doubled every generation, has been the workhorse to drive the processor development in the past decades. Nonetheless, due to the ever-widening gap between the improvement in the capability of heat spreading and the soaring transistor count, further continuance of Moore's Law for the upcoming technology nodes becomes an increasingly challenging issue. Moreover, as we enter the sub-32nm era in recent years, the approaching scaling limit of traditional transistors emerges as another threat to processor upswing, largely exacerbating the conundrum. In order to maintain the pace of processor development as expected in the next decade, material physicists and computer architects have made substantial efforts to break through the main constraints encountered during processor manufacturing. Heterogeneity is among the most attractive and successful solutions. In this work, we present a series of research studies related to the exploitation of heterogeneity, aiming to provide general principles for the design and optimization of future processors.

1.1 Why Heterogeneity is Important

In the past decades, processor manufacturers were persistent on enhancing the performance of their products by increasing the core frequency. Due to the near-cubic relation between processor frequency and power consumption, however, building faster processor easily translates to dramatic increase in the processor power and chip temperature, because all consumed electrical power is eventually radiated as heat. As a consequence, the clock frequency of a single processor cannot rise to arbitrarily high values, in order to avoid unreasonably high power consumption. This so-called "power-wall" issue, which stands as one of the most critical constraints in processor development, has led to a shift into the multi-core (e.g., chip multi-processors, or CMPs) and many-core (e.g., graphics processing units, or GPUs) design paradigm where each individual

core is designed with reasonable complexity running at a suitable frequency to encapsulate the total power consumption within a pre-set budget without violating the thermal constraint.

To date, most multi-core and many-core processors are designed in a homogeneous manner, where identical cores are integrated on the same processor die. Driven by the un-ceasing demand for higher performance, each individual core is evolving towards more sophisticated and faster operation, resulting in constant increases in the chip-level power consumption. To extend the development of the multi-core and many-core paradigm and more efficiently utilize the power resource, computer architects have recently proposed the concept of "heterogeneous architecture" as a substitute for the traditional homogeneous design pattern. Initial studies from both academia and industry demonstrate that heterogeneous architecture is effective in overcoming intrinsic drawbacks of homogeneous multi-processors and improving the execution energy efficiency. Heterogeneous architecture can be implemented in various manners and several designs have been introduced into the processor products delivered by leading chip manufacturers. For instance, the accelerated processing unit (APU) [1] from AMD combines CPU and graphics processing units (GPU) together, aiming to enhance the media processing capability with higher efficiency. The ARM big.LITTLE multiprocessor [2] implements another important design philosophy by including a cluster of powerful Cortex A15s and a set of smaller yet low-power Cortex A7s on the same chip, in order to deliver impressive energy efficiency via smart task scheduling. Examples also include the Nvidia Tegra 3 [13] processor that consists of four faster cores and a slower companion core.

The heterogeneity in processor manufacturing is not confined to architectural asymmetry. As traditional transistors are approaching their scaling limit in the deep submicron regime, new materials are introduced to replace the conventional devices for processor manufacturing, in or-

der to extend the curve extrapolated by Moore's Law in the next decade. However, while offering promising advantages such as lower leakage power or better scalability, these emerging materials manifest different drawbacks, implying that processors built with the new devices exclusively tend to pay an overhead on performance, reliability or other important design goals. In this situation, mixing diverse materials with distinctive characteristics – device heterogeneity – appears as an attractive work-around.

While the diversities in processor architectures and manufacturing devices provide the opportunities for heterogeneity exploitation, how to effectively leverage those divergences in the different stages of processor manufacturing and operation still needs in-depth investigation. For example, given a fixed system budget in terms of die area and thermal design power (TDP), identifying the architectural configurations that lead to the optimal balance among performance, power, and other important metrics remains an open question. On the other aspect, implementing a heterogeneity-aware scheduler assigning programs to the most appropriate processor core for better energy-efficiency is among the key factors to fully exploit a given heterogeneous platform. Further, in the presence of device-level heterogeneity, appropriately blending different devices in order to leverage their respective advantages during processor manufacturing are of great significance. In this work, we elaborate a series of studies to address these challenging problems in detail.

1.2 Dissertation Organization

The presented works take both architectural and device heterogeneity into consideration, demonstrating useful observations and techniques that are instructive to operations in the pre- and post-silicon stages of processors' lifespan. We first conduct a careful examination of possible architectural configurations for a heterogeneous CMP. Design space exploration is widely

considered as an indispensable step in the pre-silicon stage of processor manufacturing as it assists to elect the design options leading to good tradeoffs among multiple design goals. For the heterogeneous CMP in study, we explore its design space by varying the number of processor cores that show distinctive performance and power characteristics, in order to identify the most energy- and cost-efficient configurations. Furthermore, considering that the fundamental cause of the "power wall" is the substantial heat that cannot be dissipated in time, we also evaluate processors built with emerging low-power materials and observe that how recent breakthroughs in semiconductor technology can benefit the processor fabrication.

After investigating the pre-silicon stage, we shift our concentration to the heterogeneity-aware scheduling. Namely, given a heterogeneous CMP running multiple programs with different execution behaviors, how shall we assign those jobs to individual cores to achieve the optimal energy-efficiency? Designing such a scheduler requires us to bridge the gap between program execution behaviors and system energy consumption; therefore, we employ an advanced statistical tool, Patient Rule Induction Method (PRIM) [41], to facilitate our analysis and develop an energy-efficient scheduling algorithm for heterogeneous CMPs.

Compared to the second study which focuses on the architectural heterogeneity, our third work pays particular attention to the device heterogeneity. Specifically, we concentrate on a recently implemented transistor architecture, Fin Field-Effect-Transistor (FinFET) [7], and aim at enhancing the endurance of FinFET-made processors by utilizing the device-level heterogeneity.

To summarize, the main contributions of this research are as follows.

- We find that applying any single device material exclusively in the chip fabrication fails to produce efficient processors with regards to multiple perspectives including perfor-

mance, energy, and thermal effects. On the contrary, appropriately mixing diverse materials can fully leverage their advantages in performance and energy aspects and yield the most efficient chip.

- We explore processor designs with two-fold heterogeneity in terms of both manufacturing devices and core types, and exhibit that they are able to deliver extra benefit. Specifically, building big and small cores with different materials appears as the optimal design option.
- We develop a rule-set based scheduling algorithm for energy-efficient execution on heterogeneous CMPs. The scheduling condition is interpreted as a set of "IF-ELSE" conditions with regard to common performance metrics on involved cores. The scheduler dynamically makes decisions for program assignment by comparing the runtime execution behaviors with the selected rules at each scheduling interval. When the conditions on both cores are satisfied, the scheduler predicts that a job swap will be more energy-saving than the current mapping, thus switching the programs on the big and small cores accordingly.
- We demonstrate a hybrid-device design for future graphics processing units manufactured with FinFET. With replacing a few small hardware components with function-equivalents built with traditional transistors, the proposed design is capable of largely enhancing the device's reliability with slight performance degradation.

The remainder of the dissertation is organized as follows. We present our exploration on the design space of heterogeneous CMPs in section 2. Section 3 elaborates the energy-efficient scheduling work by introducing the employed statistical tools and algorithm development. In

section 4, we demonstrate the hybrid-device GPU design in detail. We finally summarize the studies and present potential works in section 5.

CHAPTER 2. OPTIMAL CONFIGURATION SELECTION FOR HETEROGENEOUS CMPs

2.1 Overview

Processor manufacturers have complied with Moore's Law to double the transistor count and performance on each new generation product in past decades. However, as we embrace the deep submicron era, Dennard scaling which describes the continuous decrease on the supply and threshold voltage of a transistor at each new technology node has stalled [38][83], leading to an ever increasing power density on modern processors. On the other hand, the maximum processor power consumption should be always enclosed within a reasonable envelope despite the manufacturing technology, due to physical constraints including heat dissipation and power delivery. Under this limitation, a large portion of integrated transistors on a future processor must be significantly underclocked or even completely turned off in order to satisfy the power constraint and maintain a safe working temperature. This phenomenon, which is termed "dark silicon", is recognized to be one of the most critical constraints that prevent us from obtaining commensurate performance benefit from the increased number of transistors.

Dark silicon might be exacerbated as Moore's Law continues to dominate the processor development. Figure 2-1 illustrates the scaling trend of the amount of "dark" transistors according to the ITRS roadmap [8]. As can be seen, the percentage of the dark area on a chip is exponentially expanding at each generation. This results in a chip with up to 93% of all transistors inactive in a few years from now [105]. Therefore, seeking new design dimensions to efficiently utilize the chip-level resource including power and area is important for us to obtain sustainable performance improvement in the future. Prior works have proposed a few solutions to address the dark silicon problem from certain aspects [38][48][83][103][107][108]. However, most of these works mainly concentrate on a specific solution, lacking general justifications of multiple

design options. Considering that an initial guidance to the design of future processors in the presence of dark silicon is highly desired, we conduct a comprehensive assessment of new design dimensions with special concentration on heterogeneity in the early stage of processor manufacturing.

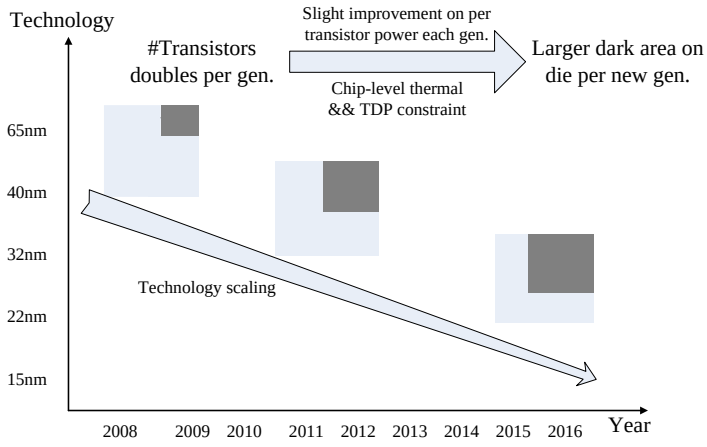


Figure 2-1. Increasing dark area with technology scaling

Our target processor is a chip multiprocessor (CMP) with fixed power and area budget. The first dimension that will be evaluated is device heterogeneity. Since dark silicon is essentially caused by the slow improvement in a CMOS device's switching power, emerging low-power materials might be used to build processors in order to illuminate the dark area. However, many power-saving devices manufactured with nano-technology manifest a series of drawbacks such as long switch delay [54]. Due to this limitation, it is inappropriate to use such devices to completely replace the traditional CMOS in processor manufacturing. To effectively alleviate the power constraint without suffering from significant performance degradation, integrating cores made of different materials on the same die emerges as an attractive design option. A few works have justified the feasibility of hybrid-device CMP at circuit level [62][92][93][102] while some of them further demonstrate the advantage of the resultant processors in performance improve-

ment [62]. Nevertheless, these works are mainly conducted on a fixed platform and thus the optimal design configuration which provides desirable balance among disparate evaluation metrics remains an open question. On the other hand, architectural heterogeneity (e.g., including both big and small cores on a processor) has been proved an effective solution to energy efficiency improvement. Therefore, jointly applying the device heterogeneity and architectural heterogeneity becomes a promising option to further exploit their advantages over conventional designs, hence the second design dimension "two-fold heterogeneity". In general, by evaluating the described new design dimensions in detail, we make the following key observations in this chapter:

- We demonstrate that using diverse materials in the chip fabrication is effective in relieving the dark silicon problem. By integrating more cores made of slower and power-saving devices and relatively few cores built with faster yet power-consuming devices, more processor cores can be enabled. Therefore, the advantages of both materials are leveraged, assisting us to produce processors that deliver impressive energy- and cost-efficiency.
- We observe that architectural heterogeneity is capable of offering higher cost-efficiency in addition to the well-known energy-efficiency over conventional designs, because including small low-power cores is able to reduce the peak chip temperature and thus decreasing the cooling expense. This further confirms the importance of building CMPs with different types of cores in the presence of dark silicon.
- We explore processor designs with two-fold heterogeneity with regards to both manufacturing devices and core architectures. We show that building complex out-of-order cores with power-saving devices while manufacturing small in-order cores with relatively power-consuming material is able to deliver extra benefit on energy- and cost-efficiency, thus appearing as the optimal design option.

2.2 Methodology

2.2.1 Heterogeneous Architecture

The heterogeneous platform considered in this chapter is a single-ISA CMP containing *m* big and *n* small cores. Note that both *m* and *n* are no less than zero. Figure 2-2 illustrates its architectural overview. As can be seen, each core is equipped with a private L1 cache, connecting to the shared L2 cache via an interconnection. The main memory stands as the lowest level in the memory hierarchy and communicates with the shared cache through a memory controller. Note that although our study is conducted on CMPs with shared last-level caches (LLC), the rule-set guided scheduling approach can also be adapted to systems without shared LLC and effectively increase the energy efficiency.

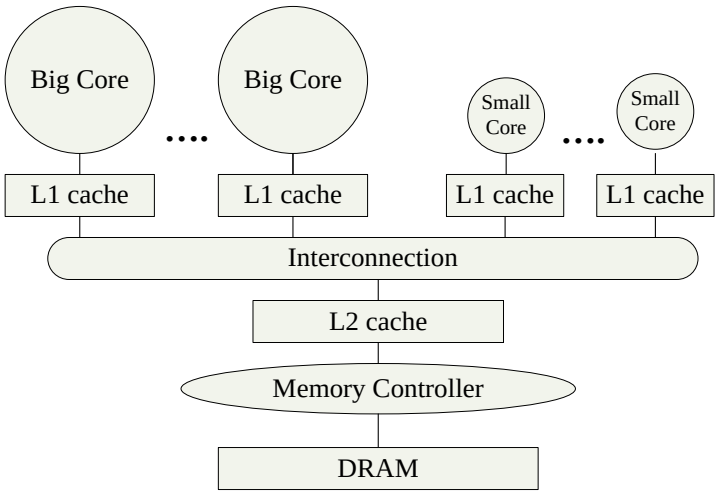


Figure 2-2. Architectural overview of the heterogeneous architecture

2.2.2 Evaluation Metrics

In this subsection, we describe the metrics for the evaluation of different configurations. Note that we characterize multiple aspects including performance, energy efficiency, thermal features and cost-efficiency for each design dimension, in order to make a comprehensive investigation.

We choose total execution time to evaluate the performance. Our study consists of investigations on both homogeneous and heterogeneous architectures, thus execution time is a preferable metric for the performance measurement. This is especially important when we run parallel applications on CMPs with varied number of cores, where the total instruction count might fluctuate due to different parallelization overhead. In this situation, other common adopted metrics such as IPC or global throughput are inappropriate for the evaluation.

For the energy-efficiency and thermal feature, we use energy-delay product (ED) and peak temperature for assessment. Besides these three extensively discussed metrics, we also include cost-efficiency as the fourth factor for investigation. In this work, we define the cost efficiency as MIPS/dollar. The considered cost is composed of the die expense and cooling cost, where the former part can be calculated with the following equations [81]:

$$Die\ cost = \frac{wafer\ cost}{Dies\ per\ wafer \times Die\ yield} \quad \text{2.1}$$

$$Dies\ per\ wafer = \frac{\pi \times \left(\frac{wafer\_diam}{2}\right)^2}{Die\ area} - \frac{\pi \times wafer\_diam}{\sqrt{2 \times Die\ area}} - Test\ dies \quad \text{2.2}$$

$$Die\ yield = wafer\ yield \times \left\{1 + \frac{Defects\ per\ unit\ area \times Die\ area}{\alpha}\right\}^{-\alpha} \quad \text{2.3}$$

Table 2-1 lists the values of referred parameters derived from recently released data in industry. The cooling cost is computed based on a model that is introduced in a prior work [115].

$$C_{cooling} = K_c t + c \quad \text{2.4}$$

In general, this cost is decided by the peak temperature achieved during the execution. Characterizing the cost-efficiency is necessary for computer architects to identify the optimal design configurations, thus deserving careful consideration.

Table 2-1. Parameter values for die cost calculation

Parameter	Value
Wafer cost	$4900
Wafer diameter	300mm
Wafer yield	0.9
Defects per unit area	0.4/cm^2
Alpha	3

2.2.3 Simulation Environment and Workloads

We use a modified SESC [86], a widely used cycle-accurate simulator for architectural study, to conduct our investigation. We choose McPat 1.0 [69] for power and area estimation and Hotspot 5.0 [16] for temperature calculation. Note that we assume a 22nm technology in this work, thus we set the system budget based on an Intel Ivy Bridge processor [3]. In specific, the area of the target chip should not exceed 100mm^2 and the maximal power consumption is 60W.

Recall that our design space includes configurations which integrate both big and small cores on the same chip. For this purpose, we assume a complex out-of-order core and a simple in-order core whose parameters are listed in Table 2-2. The L2 cache is set to 4MB in the base-line configuration, but may vary in later subsections (i.e., tradeoff study between cache and core). Table 2-3 lists the estimated area and peak power for each component on the chip. Given these conditions, the number of cores and LLC size that can be accommodated is determined by the following expressions:

$$Area\ constraint: N_b \times A_b + N_s \times A_s + S_{L2} \times A_{L2} + A_{all\ other} \leq 100$$

$$Power\ constraint: N_b \times P_b + N_s \times P_s + S_{L2} \times P_{L2} + P_{all\ other} \leq 60$$

The variables N_b, N_s and S_{L2} denote the number of big cores, number of small cores and L2 size (in MB), respectively. Constants A_b and P_b indicate the area and peak power for a big core as listed in Table 2-3. Similar interpretations apply to other symbols such as A_s and P_s.

Table 2-2. Architectural parameters for system components

Component	Parameter	Value
Big core	Pipeline type	out-of-order
	Processor width	4
	ALU/FPU	4/4
	ROB/RF	160/160
	L1I cache size	32KB
	L1D cache size	32KB
	L1 associativity	4
Small core	Pipeline type	in-order
	Processor width	1
	ALU/FPU	1/1
	L1I cache size	8KB
	L1D cache size	8KB
	L1 associativity	2
Other parameters	L2 cache size	4MB (varied in later sections)
	L2 associativity	8
	Cache block size	32B
	Technology	22nm
	Frequency	3G
	Chip area	$100mm^2$
	TDP	60W

Table 2-3. Estimated area and power for system components

Component	Peak power	Area
Big core	5.6W (High-K)	$7.6mm^2$
	4.8W (NEMS-CMOS)	
Small core	1.1W(High-K)	$1.9mm^2$
	0.8W (NEMS-CMOS)	
L2 cache	0.8W/MB	$3mm^2$/ MB
Interconnect	5W	$4mm^2$
Other components	11W	$23mm^2$

The workloads used for our exploration are based on the specific architecture in study. Multi-threaded programs are generally used for CMPs on which all cores have identical architecture (i.e., homogeneous); on the other hand, when both big and small cores are integrated, we consider that “heterogeneous” workloads are more appropriate for the investigation and thus use combinations of programs from SPEC CPU2006 [14] as a substitute. For those parallel applications,

the number of threads for execution always equals to the core count of the underlying CMP and all programs are executed till completion, in order to guarantee that the identical task is performed. We choose a total of 10 programs from SPLASH-2 [9], PARSEC [26] and ALPBench [74] for the simulation. The reason for not including other workloads is that their intrinsic characteristics (e.g., requiring 2^n threads) prohibit the execution on many configurations. As for the SPEC programs, each of them is simulated 100 million instructions after an appropriate fast-forwarding. This also ensures that identical tasks are performed across different configurations. Table 2-4 lists all selected benchmarks used in this study.

Table 2-4. Selected applications for simulation

Category	Benchmark Suite	Applications (Kernels)
Homogeneous	SPLASH-2	Barnes, FMM, Radix, Raytrace, Water-spatial, waterNS
	PARSEC	Blackscholes, Swaptions
	ALPBench	MPGDec, MPGEnc
Heterogeneous	Computation-intensive	h264, dealII, namd, spcrand, sjeng, omnetpp, gobmk, hmmer, bzip2
	Memory-intensive	mcf, libquantum, milc, leslie3d, perlbench, lbm, soplex, astar

2.3 Device Heterogeneity

2.3.1 New Device and Architectural Implication

The slight improvement in transistor power density is fundamentally caused by the physical characteristics of MOSFET [105]. Due to this limitation, it is intuitive to recognize that breakthroughs in semiconductor technology are the antidote to dark silicon in essence. Recently, substantial effort has been made on the semiconductor manufacturing process in order to further ex-

tend Moore's Law. In this work, we consider two representative solutions in this dimension, namely High-K dielectrical [5] and Nano-electro-mechanical switch (NEMS) [35][54].

High-K dielectrical refers to a device that replaces the silicon dioxide in semiconductor manufacture. The letter K stands for dielectrical constant, indicating how much charge the material can hold. High-K is capable of significantly decreasing the leakage current (i.e., < 1% of SiO_2) and has already been adopted by leading processor manufacturers [5]. Although High-K material has made impressive achievements in reducing leakage energy, it is not sufficiently scalable because this device still suffers from the MOSFET-imposed constraints. Nevertheless, as an important substitute of conventional devices in current industry, it deserves a careful evaluation.

The NEMS material, on the other hand, is a candidate for future processor development because it is built on physical switch and thus not limited by the drawbacks of MOSFET. NEMS is able to reduce the leakage current by orders of magnitude, however, it demonstrates a significantly longer switch delay compared to conventional devices, implying large performance degradation of the resultant processor. Taking this into consideration, researchers propose a hybrid device that combines NEMS and CMOS together. Dadgour et al. [35] elaborate the features of NEMS-CMOS circuits in detail and demonstrate the potential of this hybrid device in future processor manufacturing. Therefore, we consider NEMS-CMOS as an alternative material in this work. We carefully calibrate the parameters based on recent documents [5][35][54] for High-K and NEMS-CMOS and list the important features in Table 2-5.

Although the purpose of this section is not to make comparison among emerging devices, a glance at their characteristics can enlighten us on architectural innovation for the next generation

CMP. Specific to High-K and NEMS-CMOS, the latter material switches at a lower rate than the former one but offering extra saving for both dynamic and leakage energy. This implies that integrating High-K cores and NEMS-CMOS cores on the same chip would deliver a processor that works more efficiently than a CMP manufactured with an exclusive device. Keeping this in mind, we evaluate a set of design configurations, with which a portion of integrated cores are built with High-K while the remaining ones with NEMS-CMOS. We compare such mixed-device configurations with exclusive-device CMPs (i.e., all High-K cores or NEMS-CMOS cores) and aim at identifying the optimal design choice.

Table 2-5. Features of materials considered in this work

Material	Features
High-K	Reduce leakage energy to 25% of dynamic energy
NEMS-CMOS	OR gate: 30% higher delay, reducing 60% switching power
	SRAM cell: 25% higher delay, saving 85% leakage energy

2.3.2 Result Analysis

We consider two categories of CMPs to characterize the impact of device selection. The first group of chip-multiprocessors is composed of big out-of-order cores while the ratio of High-K cores over NEMS-CMOS cores is varying. Recall that the L2 cache is fixed to 4MB in this section. As a result, the total number of big cores that can be accommodated on die is either 7 or 8. The reason of the varying core count is as follows. When all cores are manufactured with High-K, the power constraint restricts the maximal number of cores to be 7 although there is enough space for an extra core; as more NEMS-CMOS cores which consume relatively lower power are integrated to replace High-K cores, the area constraint becomes the determinative factor and con-

fines the core count to be 8. On the other aspect, when all cores are small in-order ones, the core count is always limited by the area constraint and should not go beyond 30.

We run parallel applications with these configurations for evaluation. Figure 2-3 plots the average performance and energy-efficiency of these applications. All results are normalized to that corresponding to the 7H_0N configuration in the "big" category, where the chip contains 7 out-of-order cores made of High-K. Note that in later sections of this chapter, we also show results in this normalized fashion. The notation *x*H_*y*N means a total of *x* High-K cores and *y* NEMS-CMOS cores are installed. Also recall that the performance is measured in execution time, thus smaller values indicate better performance. As can be observed, in the "big" category, the execution time gradually increases at first and demonstrates a significant reduction from 4H_3N to 3H_5N, after which the curve rises again. The reason of the performance degradation (e.g., from 7H_0N to 4H_3N) is that NEMS-CMOS cores execute at a lower rate than the High-K counterparts; therefore, increasing the number of NEMS-CMOS cores inclines to prolong the overall execution time. The performance improvement at 3H_5N comes from the extra core in this configuration, with which the applications are executed with one more thread. A case study will be given shortly to further analyze this issue. As for the "small" category, the execution time gradually increases since the core count is fixed to 30 irrespective of the manufacturing device.

The energy-efficiency demonstrates a different variation from the performance change. In general, the energy-delay product is decreasing as more NEMS-CMOS cores are equipped. This is because the energy saving from NEMS-CMOS cores outweighs the corresponding performance degradation while running these parallel applications, thus using more such cores is beneficial to improving the energy-efficiency. The only exception is observed at the switch from 1H_7N to 0H_8N in the "big" category (or 2H_28N to 0H_30N in "small"), where the energy-

delay demonstrates a slight increase. This is due to the fact that the performance degradation contributes more to the variation of ED for programs with long serial phase. With the 0H_8N configuration, the sequential stages are executed on the NEMS-CMOS cores, thus resulting in significant performance loss and higher ED.

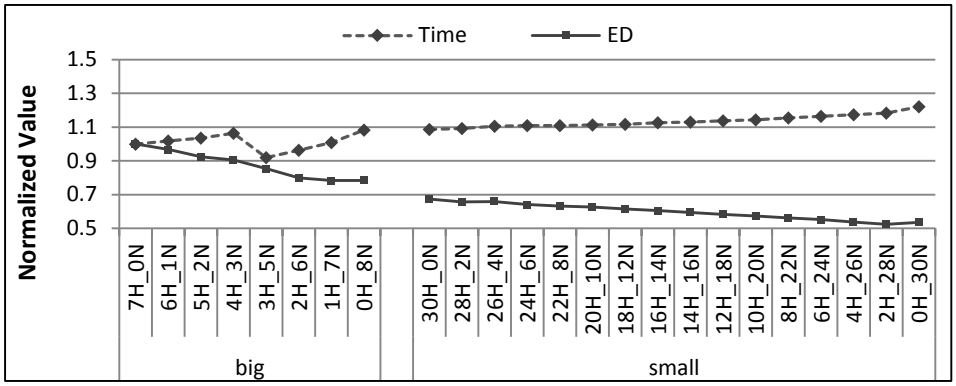
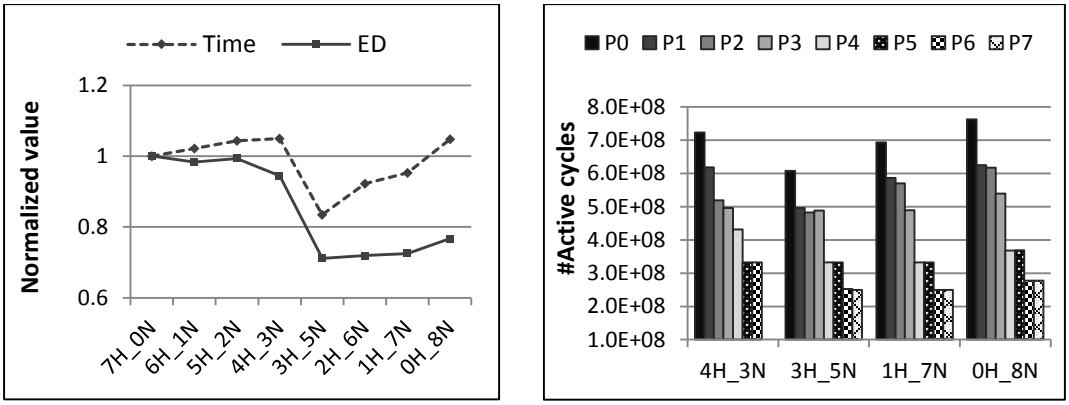


Figure 2-3. Average execution time and energy-efficiency of parallel applications running on CMPs with different materials

In summary, for a CMP which only consists of big cores, including both High-K and NEMS-CMOS cores is a preferable configuration than building the chip with a single type of cores. From the performance perspective, the 3H_5N configuration is able to shorten the execution time by an average of 8.9% while reducing the ED by 14.2% compared to the 7H_0N design. The ED-optimal configuration (i.e., 1H_7N) can save the ED by up to 20.8% with ignorable performance loss in comparison with 7H_0N. As for the small-core-oriented architecture, the best energy-efficiency is also delivered by a mixed-device CMP (2H_28N), although the optimal performance is obtained on the all High-K chip. Nevertheless, it is still reasonable to conclude that mix-device CMPs generally outperform those exclusive-device chips.

To further understand the reason of the performance scaling trend shown in Figure 2-3, we choose a representative application (*MPGEnc*) from the program set for analysis and demon-

strate the results in Figure 2-4. Note that we only show the results on CMPs with big cores. The *MPGEnc* benchmark implements a parallel version of MPEG-2 encoder. In this application, the threads are respectively forked and joined at the beginning and end of each frame. Each thread is responsible for encoding a set of macroblocks of a frame while thread 0 always operates on its dedicated buffer. The task assigned to each thread is not identical, thus the time spent by each thread also varies.



(a) Performance and ED (b) Per-core active cycles

Figure 2-4. Execution information of a selected benchmark: *MPGEnc*

Plot (a) demonstrates the performance and ED scaling while Plot (b) shows the active cycles of each core during the execution of this program with four configurations. The total execution time is determined by the main thread running on the first processor (P0), and the performance of the parallel stage can be generally estimated from the active cycles of P1. As can be observed, since the number of threads is increased from 7 to 8, the 3H_5N configuration takes much shorter time than 4H_3N to finish the encoding due to the acceleration in parallel stage, hence the remarkable performance improvement at 3H_5N. For the latter three configurations where the core counts are identical, the performance degradation is caused by the decreasing of faster cores (High-K). Specificically, the 1H_7N organization includes only one High-K core (P0) while three such cores are equipped in 3H_5N; as a consequence, the parallel stage needs longer time

to complete on the CMP configured as 1H_7N, thus lowering the overall performance. On the other hand, the performance degradation from 1H_7N to 0H_8N essentially stems from the slow execution of the sequential stage. This is especially critical for programs with long initialization and finalization.

Peak temperature and cost-efficiency are another two important metrics for comprehensive evaluation of a design configuration. We demonstrate the results of these two features for the proposed configurations in Figure 2-5. As shown in the figure, the temperature drops significantly as we employ more NEMS-CMOS big cores. The reason is that the power density on a NEMS-CMOS core is remarkably smaller than that of a High-K counterpart, thus a NEMS-CMOS core is relatively "cooler" compared to a High-K one. As more cool components are integrated on die, thermal coupling tends to be alleviated and the peak steady temperature is gradually decreased. Therefore, the coolest chip is the one where all cores are manufactured with NEMS-CMOS. On the other aspect, lower temperature results in lower cooling cost. This means that we are essentially trading off "performance" for "low cost" when we replace a NEMS-CMOS core for a High-K core. In this scenario, the cost-efficiency reaches the peak value at 1H_7N where the performance and cost can be optimally balanced. Note that the increment of cost-efficiency from 4H_3N to 3H_5N is resulted from the performance boost. The curve corresponds to the "small" category is relatively smooth. The reason is that the in-order cores consume much smaller power than big cores and thus generate less heat. Since the temperature remains relatively low, replacing the High-K small cores with NEMS-CMOS cores cannot significantly cool the chip as it does in the "big" category. As a consequence, the maximal cost-efficiency is achieved when all cores are made of High-K (30H_0N).

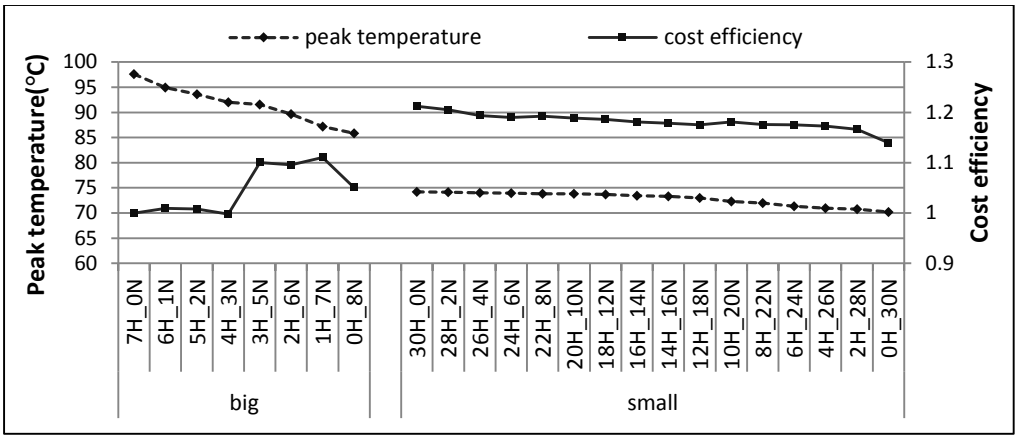


Figure 2-5. Average peak temperature and cost efficiency of multi-threaded benchmarks running on mixed-device CMPs

2.4 Architectural Heterogeneity

While processor heterogeneity can be interpreted in various manners, integrating processor cores with distinctive complexities and performance/power on a single chip appears as the most easily-implemented approach. Several products with such an organization have been introduced in recent years. For example, AMD Fusion [1] and Intel Ivy Bridge [3] integrate traditional CPU cores and graphics processing units on the same die. The big.Little [2] platform developed by ARM includes both big complex processors and small power-saving cores on the same chip. These heterogeneous architectures are expected to improve the energy-efficiency of next generation CMPs in the presence of dark silicon. In this work, we mainly focus on the second type where all cores are conventional CPUs but deviating in performance and power consumption.

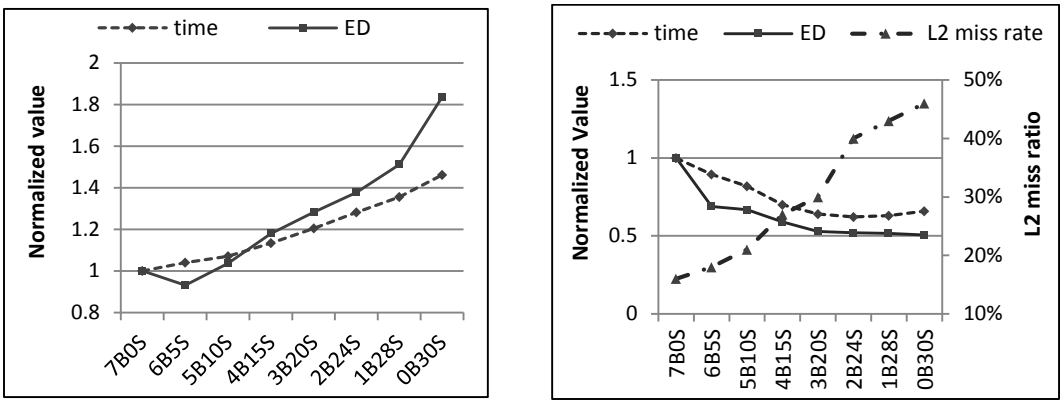
2.4.1 Performance and Energy Efficiency

We run "heterogeneous" workloads which consist of multiple programs from SPEC 2006 to investigate the architectural-heterogeneous architecture. Considering that program features such as memory intensity determine the computation efficiency on heterogeneous CMPs [48], we briefly classify the programs from SPEC CPU 2006 into two categories based on their L2 miss

ratios. Programs falling to the computation-intensive group show relatively lower cache miss rate and obtain significant performance improvement on big cores; the memory-intensive workloads, on the contrary, suffer from frequent LLC misses and get fairly limited benefit in terms of performance while executing on big cores. Since the goal of this work is to characterize and make comparison among different configurations, we consider that running these two groups of applications separately leads to a more convincing conclusion. Note that our CMP can accommodate 30 cores at most, so we always run 30 programs across all configurations while each core is executing one program at a time. In addition, we assume that all cores on chip are manufactured with an exclusive device in this section.

Figure 2-6 (a) and (b) respectively show the normalized performance and ED for computation-intensive and memory-intensive workloads running on a High-K CMP with varied configurations. The notation *x*B*y*S indicates that x big cores and y small cores are integrated on the chip. Again, the core counts are determined by both area and power constraint as described in section 2.2. From Plot (a) we observe that the total execution time of the computation-intensive workloads keeps increasing as the number of big cores is reduced. This is due to the fact that the execution speed of such programs on big cores is remarkably faster than that on small in-order cores. For example, the relative performance (i.e., time on small core/time on big core) of *dealII* is around 6.02, implying that running 6 such programs on a big core sequentially takes even shorter time than running them on 6 small cores in parallel. As a result, 7B0S is the optimal configuration from the standpoint of performance. The energy-delay product reaches the minimal value when building 6 big and 5 small cores on the chip. With this configuration, the energy-efficiency is increased by 9.1% with sacrificing 3.7% performance. As more small cores are introduced on

die after that, the ED is gradually increasing because of the significantly prolonged execution time.



(a) Computation-intensive workloads (b) Memory-intensive workloads

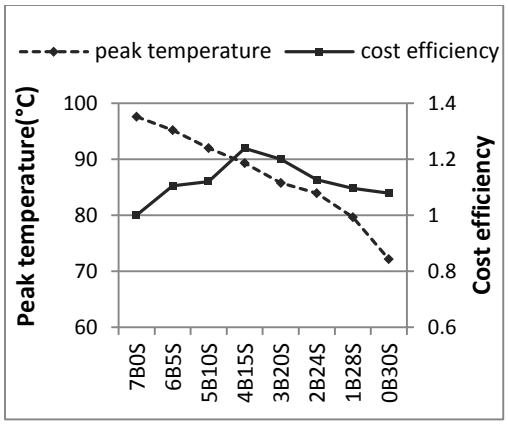
Figure 2-6. Performance and ED of benchmarks running on architectural heterogeneous CMPs with High-K devices

Memory-intensive programs demonstrate a completely distinctive preference for performance and energy-efficiency optimization. As can be observed from Plot (b), the global execution time is generally getting shorter if the total number of cores is increasing. This is because running memory-intensive workloads on big cores obtains quite limited performance benefit; therefore, using multiple small cores to replace a big core enables faster execution for the entire job. However, the performance curve almost flattens after the configuration 3B20S. This results from the intensive contention on the L2 cache. Since each program generates a large amount of L2 accesses, introducing more concurrent threads will significantly exacerbate the cache performance of individual programs. In this condition, the benefit from extra cores tends to be largely mitigated by the per-core performance degradation when the core count exceeds a certain threshold. To illustrate this, we demonstrate the variation of L2 cache miss ratios in Figure 2-6 (b). As can be seen, the miss ratio gradually increases from 16.2% at 7B0S to 45.6% at 0B30S, thus we do not see impressive performance boost after the core count goes beyond 23 (3B20S). In gen-

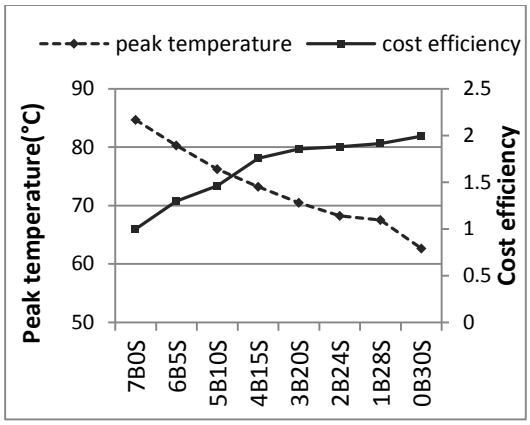
eral, the optimal configuration with respect to performance is 2B24S which only takes 63.2% of the time in 7B0S while reducing the ED by 49.2%. We also evaluate CMPs made of NEMS-CMOS and observe fairly similar trend as demonstrated in Figure 2-6; therefore, the same analysis also applies.

2.4.2 Thermal Feature and Cost Efficiency

We now shift our concentration to the thermal behavior and cost efficiency. Figure 2-7 (a) and (b) respectively plots the variations of these two metrics for computation-intensive and memory-intensive workloads running on High-K CMPs. For the former category, the temperature drastically drops as we gradually remove big cores to accommodate more small cores. This is straightforward to understand since small cores are much simpler and consume less power than big cores. The common hotspots in an out-of-order processor such as the instruction issue queue have been eliminated from small cores, thus replacing big cores with small cores is effective to decrease the chip temperature and save the cooling cost. However, computation-intensive workloads favor big cores for better performance, implying that the performance will be degraded as we reduce the number of big cores. In this situation, the interplay between performance and temperature results in a non-monotonic variation of the cost efficiency that it first increases to the peak value at 4B15S and then drops as the big core count is further decreased. Specifically, the 4B15S configuration is able to cool the chip by 7.5℃ while improving the cost-efficiency by 23.9% compared to the 7B0S organization. As for the memory-intensive workloads, since both performance and temperature prefer integrating more small cores in general, the cost-efficiency reaches its peak value at the 0B30S configuration, which decreases the chip temperature by 21.2℃ and delivers 1.99X higher cost-efficiency in comparison with 7B0S. Similar conclusion can be drawn based on the investigation of NEMS-CMOS designs.



(a) Computation-intensive workloads



(b) Memory-intensive workloads

Figure 2-7. Peak temperature and cost-efficiency of benchmarks running on architectural heterogeneous CMPs with High-K devices

2.5 Two-fold Heterogeneity

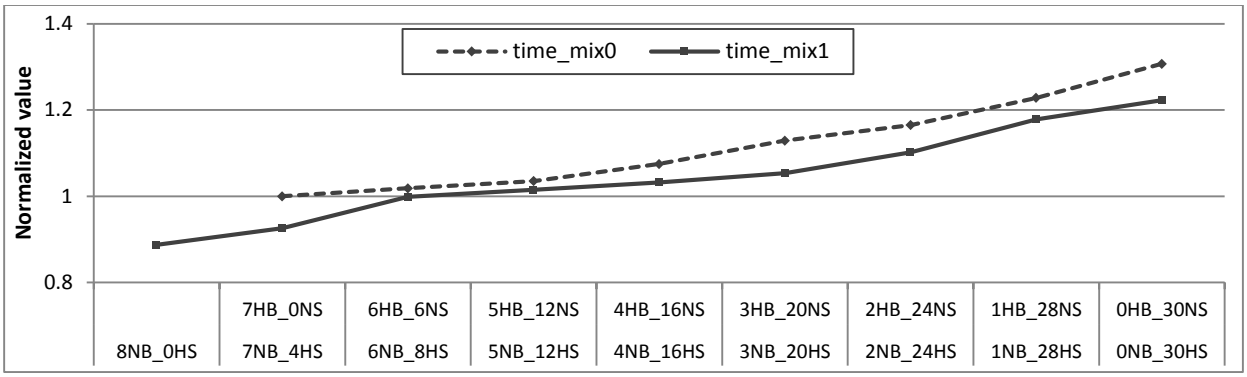
The results demonstrated in the previous two sections justify the importance of heterogeneity, respectively from the standpoint of manufacturing device and core type. Based on this observation, it is intuitive to consider blending two design dimensions to seek further improvements. In this section, we evaluate a set of configurations where both the material and complexities are different among integrated cores. We assess two kinds of organizations: big High-K cores along with small NEMS-CMOS cores and vice versa. We run both computation- and memory-intensive workloads for the evaluation. As we will demonstrate in later sections, two-fold heterogeneity designs can benefit both types of workloads.

2.5.1 Performance and Energy Efficiency

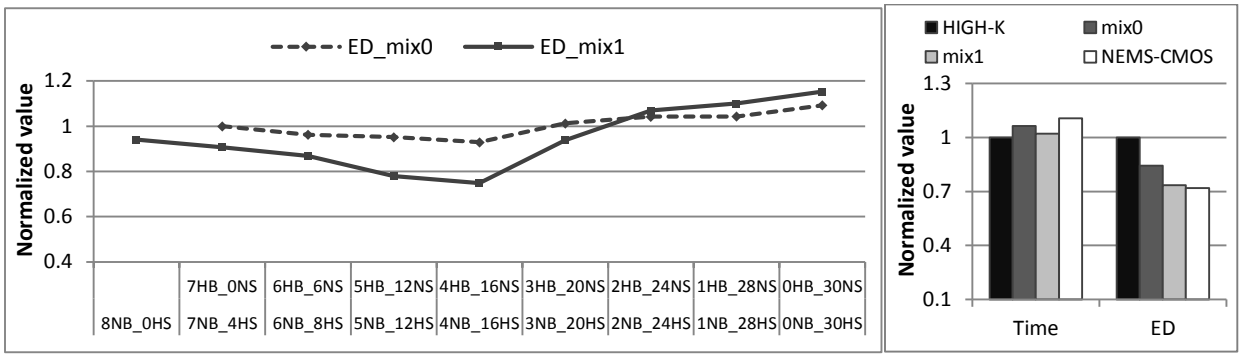
Figure 2-8 (a) plots the performance scaling of computation-intensive programs with these two design patterns. Note that all results are normalized to that in the 7HB_0NS case. The upper labels on the horizontal axis correspond to the first architecture where big cores are made of High-K and small cores are manufactured with NEMS-CMOS (mix0 or *x*HB_*y*NS); accordingly, the lower labels correspond to the opposite architecture which includes big NEMS-CMOS and

small High-K processors (mix1 or *x*NB_*y*HS). As can be observed, configurations with the second pattern, namely *x*NB_*y*HS, always outperform the counterparts from the first category. This can be explained in two aspects. First, since NEMS-CMOS cores are relatively power-saving, the second design pattern accommodates more processors when the core count is power-limited. Due to this reason, the total number of cores is larger in the *x*NB_*y*HS designs, thus these configurations take shorter time to finish executing the program combination. This corresponds to the scenarios where the number of big cores is no smaller than 6. Second, as the constraint factor shifts to chip area, the core counts in both design patterns become identical (from 5B_12S). In this situation, the global execution time basically depends on the performance of small cores because of their larger amounts. For instance, in the 2B_24S configuration, how fast the programs run on small cores determines the overall performance in essence, because the number of small cores is 11 times larger than that of big cores. Since those in-order processors are made of High-K, the chips designed with the second pattern still offer better performance.

Figure 2-8 (b) demonstrates the variation of the energy-efficiency for the same program set running with considered configurations. Note that the interplay between the performance/energy of different cores makes the variation of ED non-monotonic. For both blending patterns, we note that the energy-delay product gradually decreases at first until the minimal value is reached at 4B_16S, after which the efficiency is getting worse. More specifically, the *x*NB_*y*HS delivers better energy-efficiency than the *x*HB_*y*NS when the configuration is varied from 8 big cores to 3 big cores. This is due to the shorter execution time and less energy consumption on big NEMS-CMOS cores. As small cores begin dominating the chip in 2B_24S and beyond, their relatively large energy consumptions mitigate the performance benefit and make the ED rise again.



(a) Performance



(b) Energy-delay product (c) Comparison between patterns

Figure 2-8. Performance and ED of computation-intensive benchmarks running on different mixed-device CMPs

To more clearly illustrate the benefit of such two-fold heterogeneity, we identify the most energy-efficient configurations for four different design patterns, namely High-K for all cores, *x*HB_*y*NS (mix0), *x*NB_*y*HS (mix1) and NEMS-CMOS for all cores, and make comparison between these material-dependent optima. Based on the discussion in section 2.4.1, we choose 6B_5S and 6B_8S for High-K and NEMS-CMOS, respectively. We then select 4B_16S for HB_NS and NB_HS based on Figure 2-8 (b). We normalize the execution time and ED to those corresponding to the 6B_5S High-K processor and demonstrate the results in Figure 2-8 (c). As can be observed, the CMP with 4 NEMS-CMOS big cores and 16 High-K small cores

(4NB_16HS) is the global optimal configuration. It improves the energy-efficiency by 27% with only 4.3% performance degradation compared to the optimal High-K CMP.

The results of memory-intensive workloads are slightly different. We observe that the optimal configurations for mix0, mix1, and NEMS-CMOS lead to fairly close execution behaviors. This is because programs with intensive memory accesses are less sensitive to core speed compared to computation-intensive applications. In general, the optimal configurations from the aforementioned three categories all deliver 18% reduction in ED with less than 4% performance loss. As a consequence, the *x*NB_*y*HS design pattern (i.e., mix1) is the most energy-efficient for a general-purpose CMP since it is amiable to both computation-intensive and memory-intensive workloads.

2.5.2 Thermal Feature and Cost Efficiency

Figure 2-9 plots the peak temperature and cost-efficiency of these two-fold heterogeneous CMPs running computation-intensive programs. As we have observed previously, NEMS-CMOS cores result in lower temperature than High-K cores and small cores are much cooler than big ones. Consequently, the second design pattern (i.e., *x*NB_*y*HS) inclines to be cooler than its alternative (*x*HB_*y*NS), because the hotspot on die which is usually located in the out-of-order processor has lower temperature. Recall that the *x*NB_*y*HS also delivers better performance. Therefore, its cost-efficiency is significantly higher than that offered by *x*HB_*y*NS configurations. As can be seen, for computation-intensive workloads, the cost-efficiency reaches the peak value at 7NB_4HS configuration, which improves the efficiency by 20.9% compared to the 7HB_0NS case. For memory-intensive workloads, the optimal cost-efficiency is delivered by 0NB_30HS, which outperforms the baseline case by up to 66.7%. In general, we can conclude that the xNB_yHS design pattern is the optimal among all evaluated configurations.

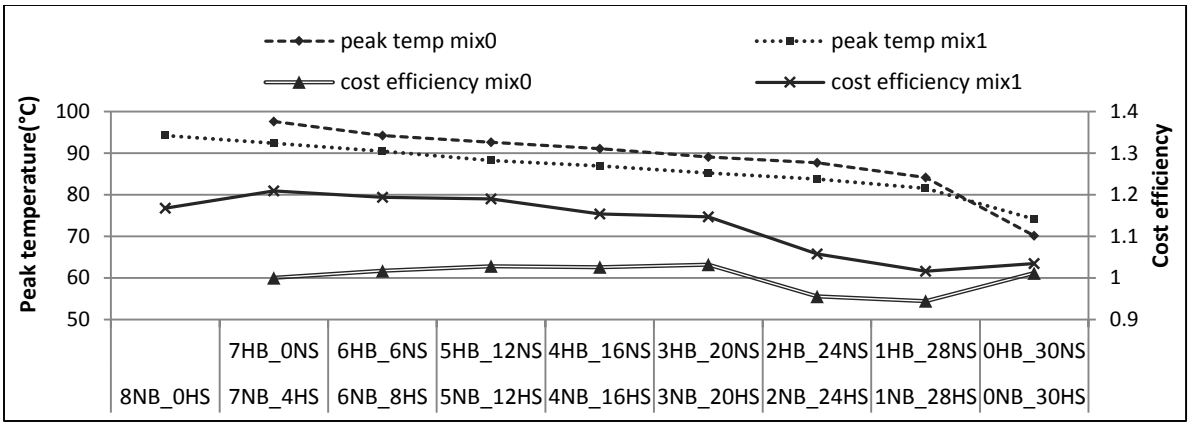


Figure 2-9. Peak temperature and cost-efficiency of computation-intensive workloads running on mixed-device heterogeneous CMPs

2.6 Extension to Varying Last-level Cache

The breakdown of area and power resources between processor cores and shared last-level cache (LLC) is also widely evaluated in energy-efficient processor design. In this subsection, we evaluate a set of configurations that integrate both big and small cores while varying the LLC size. By conducting this study, we aim at identifying the globally optimal configurations which deliver the best performance and efficiency. Note that similar to the previous section, we assume the chip is manufactured with an exclusive device and only the High-K results are demonstrated.

We generally change the core configurations with three different cache sizes, namely 4MB, 6MB and 8MB. Under this setting, we are able to configure the CMP with more core combinations. Figure 2-10 shows the performance and energy-delay product for computation-intensive workloads running on evaluated CMPs. We observe that the optimal configuration in the 4MB category also appears as the best design option across all evaluated configurations. Specifically, for computation-intensive programs, 7B0S yields the optimal performance while 6B5S delivering the highest energy-efficiency. Similarly, 2B24S and 0B30S within the 4MB category are globally optimal for the performance and ED of memory-intensive applications. This is con-

sistent with the conclusion made in the previous section that processor cores pose more impact on the overall performance and energy-efficiency than caches.

Figure 2-11 plots the variation of peak temperature and cost-efficiency with the same set of configurations. Again, the optimal design option corresponding to 4MB L2 outperforms all other candidates with respect to cost-efficiency. From the standpoint of thermal behavior, the configurations within the 4MB category lead to higher temperature than those with the 6MB and 8MB LLC, which is due to the smaller area for heat dissipation.

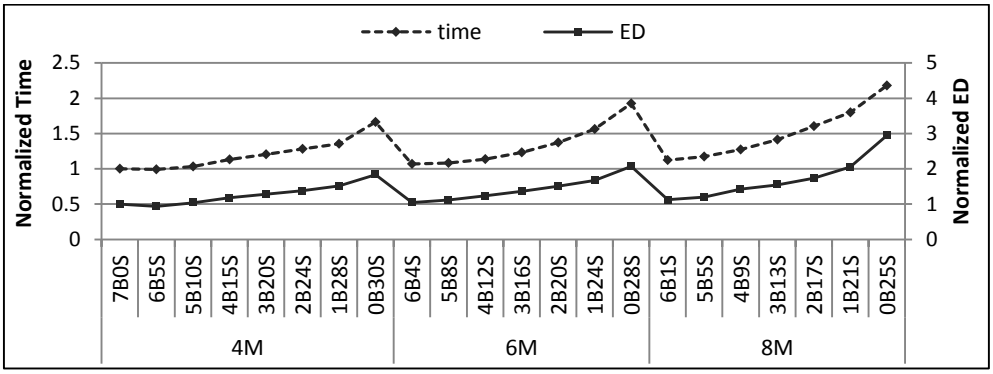


Figure 2-10. Performance and ED of computation-intensive workloads running on two-fold heterogeneous CMPs with varying LLC size

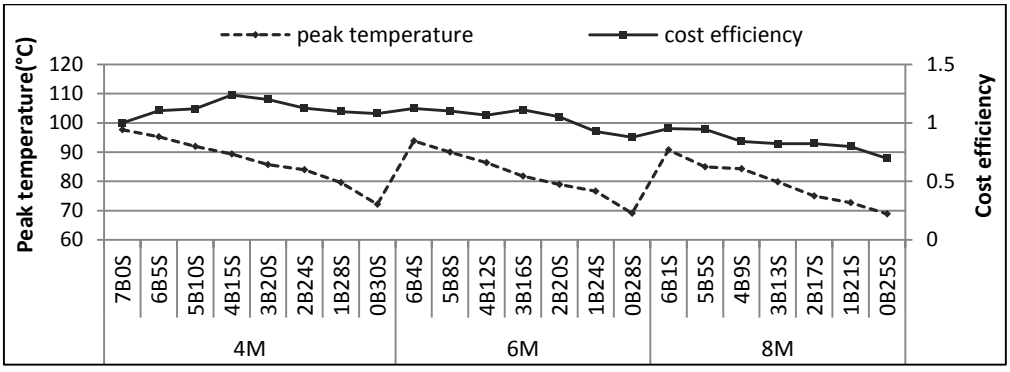


Figure 2-11. Peak temperature and cost-efficiency of computation-intensive workloads running on two-fold heterogeneous CMPs with varying LLC size

We also investigate the two-fold heterogeneity with varying LLC size, namely device heterogeneity × architectural heterogeneity × varying cache. Our experiment shows that the optimal configuration with the 4MB category still works as the most efficient globally.

2.7 Related Work

Dark silicon emerges as an increasingly important issue that menaces the scaling of Moore's Law in the deep submicron era and beyond. Due to this reason, researchers recently started to investigate this problem and proposed several solutions to alleviate the conundrum. A group from UCSD has made significant progress on using dark silicon for processor improvement. They develop conservation cores [107] and Quasi-specific cores [108] for increasing the computation energy-efficiency in different scenarios. They also introduce a platform named GreenDroid [103] to steer the design of mobile processors in the presence of dark silicon. Moore and Bailey [78] propose a similar design and implement a chip with hundreds of small nodes. Specialized architectures that aim at reducing energy consumption are also widely considered to address dark silicon. Examples include the dark silicon accelerators 0 presented by Kocberber et al. The authors design a novel architecture named indexing widget to serve domains that mainly run pointer-intensive database applications. The proposed architecture is able to reduce the energy per quest by up to 65% and effectively relieve the dark silicon. In [48], Gupta et al. demonstrate the potential of heterogeneous CMP for energy-efficiency improvement. Systems built with near-threshold voltage processors (NTV) [37][110] are also effective approaches.

Another set of solutions, on the other hand, make unique use of the existing processors instead of bringing in architectural innovations to mitigate dark silicon. For instance, Raghavan et al. [83] propose the concept of "computational sprinting", with which a chip can temporarily go beyond its thermal power budget for a short period and then return to normal status. This is ef-

fective to improve the instantaneous throughput of an application and is thereby an attractive design option for mobile systems where sustainable high performance is unnecessary. Esmaeilzadeh et al. [38] address the dark silicon challenges with approximate computing. They demonstrate that perfect precision is not always indispensable in many applications; therefore, by approximating the execution in a reasonable fashion, the energy consumption is significantly reduced while fairly high accuracy can be maintained.

While most of these studies focus on a single solution individually, few works make attempt to address the dark silicon problem from a broader perspective. Esmaeilzadeh et al. [39] use an analytical model to predict the processor scaling for the next few generations. They demonstrate that dark silicon will be heavily exacerbated as manufacture technology keeps shrinking. Based on this observation, they compare conservative scaling and ITRS scaling and show the performance potential for future processors. Taylor [105] reviews the current status of dark silicon and briefly describes four solutions from the high level. Hardavellas et al. [50] pay specific attention to the server processors and perform an exploration of design configurations.

On the other hand, studies that concentrate on early stage design with fixed system budget can be found in literature [53][73][77][116]. For example, Kumar et al. [64] investigate the design of quad-core heterogeneous chip multiprocessors and identify the optimal configurations under different power and area constraints.

2.8 Conclusion

As dark silicon has begun to hazard the scaling of Moore's Law and prohibits us benefiting from the increasing transistors as expected, new design technologies are in high demand to address this problem, in order to efficiently utilize the system resource. This is especially important

in the early stage of processor manufacturing where issues such as architectural organization and device selections need to be carefully considered. For this purpose, our work evaluates four practical design dimensions by making thorough assessments from the perspective of performance, energy-efficiency, peak temperature and cost-efficiency. We demonstrate that device- and architecture-heterogeneity are effective in using the dark area, thus yielding the optimal processors. We believe that our observations provide insightful guidance to the design of future processors.

CHAPTER 3. RULE-SET GUIDED ENERGY EFFICIENT SCHEDULING ON SINGLE-ISA HETEROGENEOUS CMPS

3.1 Motivation and Overview

In a single-ISA heterogeneous CMP, big cores are usually equipped with complex out-of-order issue logic and more function units, thus delivering superior performance compared to small cores, where executions are driven by simple in-order pipelines. Put another way, big cores accelerate the executions by consuming more power while the small cores enables power-saving at the expense of performance degradation. To more effectively utilize the core heterogeneity and leverage their respective advantages, an appropriate job scheduler that is responsible for program-to-core mapping is in high demand. In prior works addressing this problem, the program relative performance between big and small cores is widely adopted as the heuristic to guide the runtime scheduling [24][33][63][100]. Specifically, programs that gain more benefit from the execution on faster cores are selected to run on big cores; on the contrary, programs demonstrating moderate performance improvement on big cores are chosen to execute on small cores. The effectiveness of such schemes is generally dependent on the characteristics of workloads. For instance, for a two-program workload, if one shows remarkable performance boost on the big core while the other obtains quite limited benefit, a good scheduler is able to significantly improve the overall performance, thus such a workload is considered as scheduling-sensitive [33]. In contrast, if two programs exhibit similar relative performance, the workload inclines to be scheduling-insensitive.

While those proposed scheduling strategies are capable of improving performance, they do not necessarily lead to the most energy-efficient execution all the time. We use the co-execution of programs *bzip2* and *vpr* on a dual-core heterogeneous CMP as an example to justify this ar-

gument. As listed in Table 3-1, these two programs demonstrate fairly similar performance ratio between big and small cores. Therefore, by employing an existing heterogeneity-aware scheduler based on relative performance, it is likely that 1) they are randomly mapped to different cores since the scheduler recognizes this workload as scheduling-insensitive, or 2) *bzip2* is assigned to the big core as it demonstrates slightly higher performance gain than its co-runner. In Figure 3-1 we illustrate the energy consumption and energy-delay product (ED) for two possible scheduling, namely *bzip2_B+vpr_S* and *vpr_B+bzip2_S*, where the former one indicates that *bzip2* is running on the big core and *vpr* is on the small core. Similarly, the latter notation corresponds to the opposite scheduling decision. As can be observed, the second scheduling (i.e., *vpr_B+bzip2_S*) turns out to be more energy-efficient than its alternative due to the significant power reduction on the big core. Two implications can be noticed from this example. First, scheduling aiming to minimize the energy consumption and ED does not always reach consensus with the performance-oriented scheduling in a heterogeneous system. Second, for scheduling-insensitive workloads where programs have fairly close speedup on the big core, there is still plenty of room for energy efficiency optimization.

Table 3-1. Power and performance ratio of *bzip2* and *vpr*

Program	**Big core power(W)**	**Small core power (W)**	**Performance ratio**
bzip2	24.64	9.18	2.51
vpr	18.97	10.32	2.48

In this situation, seeking a new scheduling algorithm which can best exploit the energy efficiency appears more attractive in the era of heterogeneous computing. Therefore in this work, we propose a rule-set guided scheduling strategy to minimize the energy consumption for workloads running on a heterogeneous CMP. Meanwhile, our scheduler is able to deliver comparable performance to the optimal existing heterogeneous scheduler, thus achieving higher energy efficien-

cy than previous schemes. We employ an advanced statistical tool to facilitate the development of our algorithm. The tool is able to generate a set of "IF-ELSE" conditions with regard to common performance metrics on involved cores. Each condition is expressed as an inequality such as "$X_i \leqslant$ (or $\geqslant$) N", where X_i is an easily measured performance metric and N is a certain value. The scheduler then dynamically makes decisions for program assignment by comparing the runtime execution behaviors with the selected rules at each scheduling interval. When the conditions on both cores are satisfied, the scheduler predicts that a job swap will be more energy-saving than the current mapping, thus switching the programs on the big and small cores accordingly.

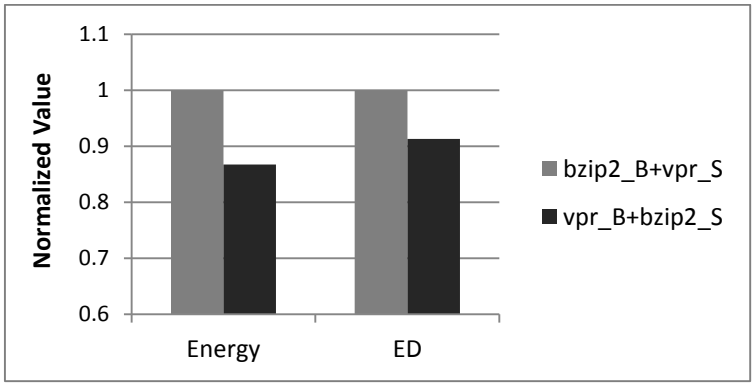


Figure 3-1. Energy and ED for *bzip2+vpr* with different mappings

3.2 Statistical Tools

As described in section 3.1, the proposed scheduling scheme is built on the measurements of common performance metrics. This introduces two challenging problems to our study. First, we should identify the important factors which impose relatively large impact on the overall energy consumption. Second, we need to quantitatively formulate the scheduling condition with regard to the selective performance metrics. Taking these into consideration, we employ an advanced statistical tool to facilitate the rule extractions and introduce the technique in this section.

3.2.1 Patient Rule Induction Method (PRIM)

In this study, we employ the Patient Rule Induction Method (PRIM) [41] to bridge the gap between program execution behaviors and the scheduling condition. PRIM is naturally suitable to facilitate this study since its objective is to find a region in the input space that gives relatively high values for the output response. The selected region (or "box") is described in an interpretable form involving a set of "rules" depicted as $= \bigcap_{j=1}^{p}(x_j \in s_j)$, where x_j represents the jth input variable and s_j is a subset of all possible values of the jth variable.

As shown by Figure 3-2, the construction of the selected region is composed of two phases: (1) patient successive top-down peeling process; (2) bottom-up recursive pasting process. The top-down peeling starts from the entire space (box B) that covers all the data. At each iteration, a small subbox b within the current box B is removed, which yields the largest output mean value in the result box B-b. We perform this operation iteratively and stop when the support of the current box B is below a chosen threshold β, which is actually the proportion of the intervals suitable for job swaps.

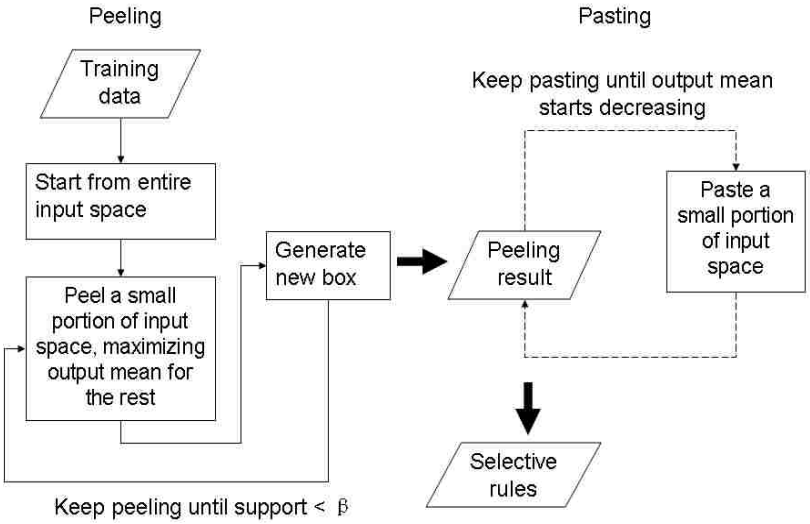


Figure 3-2. PRIM training procedure with peeling and pasting

The pasting algorithm works inversely from the peeling results and the final box can sometimes be improved by readjusting its boundaries. Starting with the peeling solution, the current box *B* is iteratively enlarged by pasting onto it a small subbox that maximizes the output mean in the new (larger) box. The bottom-up pasting is iteratively applied, successively enlarging the current box, until the addition of the next subbox causes the output mean to begin to decrease.

An advantage of PRIM over greedy methods such as tree-based methods is its patience. For example, a binary tree rapidly fragments the data because of the binary splits in that tree, while the PRIM model only peels off a small proportion of data every time. As a consequence, the solution of PRIM (hyper-boxes) is usually much more stable than those obtained from tree models.

3.2.2 Classification and Regression Tree (CART)

Although the described statistical technique PRIM is able to build a rigorous correlation between multiple input variables and a response, the accuracy of the model depends on features of the applications in the training set. Let us assume that the execution behaviors of a few intervals significantly deviate from those of other training instances while their response values are identical. In this scenario, a single universal PRIM model may not be capable of capturing all those runtime variations. This is because that the PRIM algorithm is prone to build a model that fits the majority situations in the training instances. As a result, the established model might ignore those samples appearing less frequently. Considering the diversity of program characteristics, this limitation might significantly decrease the prediction accuracy when the model is applied to different program phases or applications that demonstrate completely distinct execution behaviors to training samples.

In order to figure out this problem, we propose to partition the entire data set into several categories, each of which contains instances demonstrating similar characteristics. If we train a PRIM model for each data subset and generate a group of rule sets correspondingly, the obtained rules are supposed to be more robust and be effective to handle different execution scenarios. To achieve this goal, we employ another statistical tool named Classification and Regression Tree (CART) [27] for the data segmentation. CART has been in use for about 25 years and remains a popular data analysis tool. It provides an alternative to linear and additive models for regression problems. The CART models are fitted by a recursive partitioning whereby a dataset is successively split into increasingly homogenous subsets until the information gained by additional splits is not outweighed by the additional complexity due to the tree's growth. Trees are adept at capturing non-additive behavior, e.g. interactions among input variables are routinely and automatically handled. Further, regression tree analysis can easily handle a mix of numeric and categorical input variables.

3.3 Rule-set Guided Scheduling

While most modern operating systems support concurrent execution of multiple programs running on different cores, it is not necessary to keep all processor cores active during the entire execution. That is, a portion of cores might be idle (or in power-saving mode) while others are running in order to reduce the total energy consumption. This implies two scheduling circumstances that need to be carefully considered on a heterogeneous CMP platform; namely, (1) choosing appropriate cores to execute the programs while making other cores idle and (2) identifying a suitable task-to-core mapping when all cores need to be utilized. In this section, we will present how the rule-set guided scheduling strategy would be applied in these two scenarios in detail. Moreover, the scheduling policy should be sufficiently scalable as heterogeneous CMPs

might be configured in different manners. Therefore, we also discuss the effectiveness of the proposed strategy on heterogeneous platforms with different configurations.

3.3.1 Scheduling in Presence of Idle Cores

It is fairly common that a portion of integrated cores on a CMP are idled at runtime for the sake of power saving. For instance, assume a single-thread program is to be executed on a heterogeneous chip multi-processor similar to the big.Little platform from ARM which consists of a powerful big core and a slow small core [2]. In this situation, it makes no sense to enable both cores since one core is sufficient to run the program at any instant during the execution. Considering the representativeness of this scenario in practice, we demonstrate the single-program scheduling on a dual-core system to exemplify the implementation of the rule-set guided strategy in presence of idle cores.

For a scheduling interval, we must identify whether to run the program on the big core or the small core. Clearly, an oracle scheduler will examine these two cases during runtime (at each scheduling point) and choose the most suitable core for execution to achieve the optimal energy efficiency. However, dynamically determining the optimal schedule for a program at runtime is a challenging problem. To overcome this conundrum, we employ Patient Rule Induction Method (PRIM) to generate some selective rules on a number of performance measurements. In a scheduling interval, if the measured performance counters conform to these rules, the scheduler will map the program to the appropriate core accordingly.

More specifically, the PRIM model training is composed of the following steps. First, we select a number of typical programs for extracting the rules. For each of them, we respectively affinitize it to the big core and the small core for execution and collect a set of easily measured

performance metrics along with the energy consumption for every interval, whose length is set to a reasonable value for the study. By doing so, we can obtain the following information from the two types of cores:

Information from big core: $< X_b^1, X_b^2, X_b^3, \ldots\ldots, X_b^m, E_b >$

Information from small core: $< X_s^1, X_s^2, X_s^3, \ldots\ldots, X_s^n, E_s >$

In the tuple listed above, the *X* variables denote the measured performance counters such as the number of cache misses and the number of branch mispredictions. The subscript of each variable indicates the corresponding platform (i.e., *b* = big core, *s* = small core). The variable *E* represents the energy consumption of this interval. In this example, we measure *m* performance counters on the big core and *n* counters on the small core. Second, we compare the energy consumption for each interval under these two assignments and set a Boolean flag based on the comparison result. The flag is then used as the output of a training instance. Finally, we feed the training samples measured from all selected programs to establish PRIM models and extract the rules.

It should be noted that separate models should be established for big and small cores. This is because the program is running on either the big core or the small core at any interval, requiring two groups of conditions to respectively guide the big-to-small and small-to-big migration. Let us first focus on the big-core model that is used to manage the big-to-small migration. We assume that *E_s* is smaller than *E_b* for a specific interval. With this assumption, the training sample corresponding to this interval is $< X_b^1, X_b^2, X_b^3, \ldots\ldots, X_b^m,$ *flag*> where the flag is set to 1, indicating that the program should be assigned to the small core for energy saving. In contrast, if *E_s* is greater than *E_b*, a flag of 0 will be set. This means that running the program on the big

core is preferable (i.e., it is unnecessary to transfer the job to the small core). We train a model for the small core to govern the small-to-big migration in a similar fashion. Specifically, training instances in a form of < X_s^1, X_s^2, X_s^3, ……., X_s^n, *flag* > are fed into the PRIM tool. Note that there are two approaches to measure the runtime energy consumption in practice.

(1) If the processor provides a hardware counter to report the power usage, we just need to compare the energy consumption between the aforementioned two cases. Then we set the flag based on the comparison result. Some recently released processors such as Intel Sandy Bridge architectures and later products support dynamic power measurement by using a model-specific register (MSR) [6].

(2) In case that there is no dynamic energy reporting function on the chip, we can have an accurate estimation of runtime energy via multiplying the average power and the execution time. The dynamic power of the chip can be estimated from performance counters through another predictive model [57]. Specifically, the chip power can be added up from each component's power derived from their accessing rates, a scaling factor, and the maximal component power, plus idle power. The access rate of a component can be read and calculated from performance counters; the maximal power of each component and the scaling factors are generated and tuned by running a set of stress benchmarks.

Recall that PRIM rules identify the input space subregion that has the highest response values. Therefore, the generated rules quantify the situations that a program migration from the big core to the small core (or the other way around) is needed to achieve better energy efficiency. The selective PRIM rules are then engaged by the operating system to guide the scheduling of the program between two cores. Assume the program is randomly mapped to a core (either the

big one or the small one) initially. At a scheduling point, the performance measurements are compared with the extracted PRIM rules corresponding to the current used core. If conditions are satisfied, the model predicts that transferring the job to the other core will lead to better energy efficiency; otherwise the present scheduling is preserved. The scheduler then makes the assignment based on the prediction result and continues the execution to the next scheduling point. Note that the rule-set guided scheduling is sufficiently flexible to manage the program execution for optimizing different metrics. For instance, by changing the objective during the model construction, this approach can be easily applied to guide the scheduling in a system where performance maximization is the prime concern. Nevertheless, our concentration in this chapter is energy minimization.

3.3.2 Scheduling without Idle Cores

When the number of concurrent programs is increasing, all integrated cores on a CMP might be utilized to maximally exploit the processor computation capability. In this situation, the scheduling problem is essentially to identify the task-to-core mapping which results in the minimal energy consumption. Without loss of generality, we consider a scenario where two programs (A and B) run on a dual-core CMP consisting of one big core and one small core. For a scheduling interval, we need to compare the total energy consumption of the following two cases: (1) A on the big core and B on the small core; and (2) B on the big core and A on the small core. Between these two schedules, we should choose the one with the lower energy consumption. Similarly, we adopt the PRIM tool to generate a set of rules to guide the scheduling.

The training procedure is fairly close to that described in the previous subsection. The most significant difference lies in that a unified model regarding the performance metrics from both the big and small cores is built, meaning that conditions on big and small cores are checked sim-

ultaneously at a scheduling point. This is because that both the big and small cores are utilized to run programs, thus the execution behaviors from both sides should be monitored in order to evaluate whether a job swap leads to less energy consumption. The specific training process is as follows. First, we randomly select a certain number of program pairs. For each program pair (A, B), we assume that A runs on the big core and B runs on the small core. For each interval, we can obtain the following information by executing A and B on the big and small cores, respectively.

$$\text{Program A: } < X_b^1, X_b^2, X_b^3, \ldots\ldots, X_b^m >$$

$$\text{Program B: } < X_s^1, X_s^2, X_s^3, \ldots\ldots, X_s^n >$$

Similarly, the variables *X* denote the measured performance counters. Second, we compare the energy consumption of this schedule with its counterpart (re-running *B* on the big core and *A* on the small core), setting a Boolean variable (flag) to one if swapping these two programs will generate lower energy. Consequently, we can form a PRIM training sample by combining the above information:

$$< X_b^1, X_b^2, X_b^3, \ldots\ldots, X_b^m, X_s^1, X_s^2, X_s^3, \ldots\ldots, X_s^n, flag >$$

For each training instance, the inputs are the *m*+*n* performance counters from both cores while the output is a flag indicating if these two programs need to be switched in the next interval. We then feed all instances into PRIM to generate the conditions.

Figure 3-3 illustrates how the rule set interacts with the OS and makes decision for program assignment at runtime. The two programs are first executed on two cores (one big and one small) for an interval, respectively. At a scheduling point, the performance measurements of the current

interval are compared with the extracted PRIM rules. If conditions on both cores are satisfied, the model predicts that swapping the two programs will lead to better energy efficiency; otherwise the present scheduling is preserved. The scheduler then makes the assignment based on the prediction result and continues the execution to the next scheduling point.

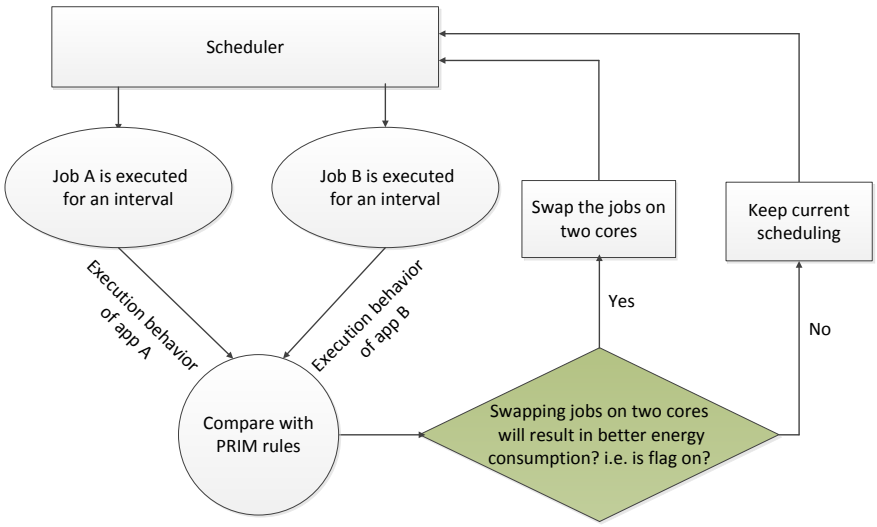


Figure 3-3. PRIM rules guided scheduling for dual-program execution

As described in section 3.2, the effectiveness of the rule guided scheduling is largely determined by the features of the programs in the training set. In cases where the programs for validation demonstrate significantly different execution behaviors from the training programs, the derived rules may not be effective in identifying the swapping cases. In this situation, the model accuracy can be further improved by preprocessing the training data. Instead of training a single PRIM model, we can build a number of different PRIM models according to the similarity of different training samples. Specifically, we use the CART mechanism to partition the input space into a few subregions. The points belonging to each individual subregion are similar in terms of energy efficiency. After that, we build a separate PRIM model for each of these subregions. Consequently, we will have a group of rule sets. When making predictions during runtime, we first

identify in which subregion the current input sample is located, then use the corresponding rule set to determine if a program switch is needed. In practice, the number of subregions doesn't need to be large. Our experiments show that partitioning the input space into 4 subregions (and also training 4 PRIM models accordingly) can result in prediction accuracy within only 5% difference from the oracle scheduler. This approach is termed Hierarchical PRIM (or H-PRIM).

3.3.3 Algorithm Scalability

Our approach is sufficiently scalable to be adopted by a system with more than 2 cores. In this subsection, we consider two generalized heterogeneous platforms and show that how the rule-set based schedulers lead to energy-efficient execution on these architectures.

We first assume a CMP with an equivalent number of big and small cores while the core count of each processor type is n. In this scenario, the optimal energy efficiency can be achieved by performing n iterations of parallel pair comparison. The scheduling process is illustrated in Figure 3-4. As shown in the figure, in the first iteration, a big core with the index i ($i \in [0, n\text{-}1]$) is compared with the small core whose index is ($n + i\%n$). All n pairs of comparisons are performed in parallel. In the second iteration, the big core i will form a group with the small core ($n+(1+i)\%n$) and make comparison correspondingly. Similarly, the comparison will be conducted between the big core i and the small core ($n+(n\text{-}1+i)\%n$) in the nth iteration. Note that the mod operations are involved to emulate the rotational comparisons. We prove that this method will lead to the optimal scheduling as follows.

Since we have n big cores and n small cores, as well as $2n$ jobs running on them, the optimal schedule is a situation that n jobs suitable running on the big cores for low energy consumption (we label the jobs as "1"s) will be assigned to n big cores and the remaining n jobs, denoted as

"0"s, will be allocated on *n* small cores. We claim that all "1" programs will be assigned to big cores and all "0" jobs will be allocated on small cores after *n* iterations, even though we are unaware of the program classification at the beginning, i.e., whether a program belongs to "1" category or "0" category. During each of the *n* iterations, we have *n* parallel comparisons between big and small cores. For each comparison, we seek better energy efficiency for two programs running on a big-small core pair. Therefore, we have four possible situations before the comparison:

(1) a "1" job running on a big core compared with a "0" job running on a small core;

(2) a "0" job running on a big core compared with a "1" job running on a small core;

(3) a "1" job running on a big core compared with another "1" job running on a small core;

(4) a "0" job running on a big core compared with another "0" job running on a small core.

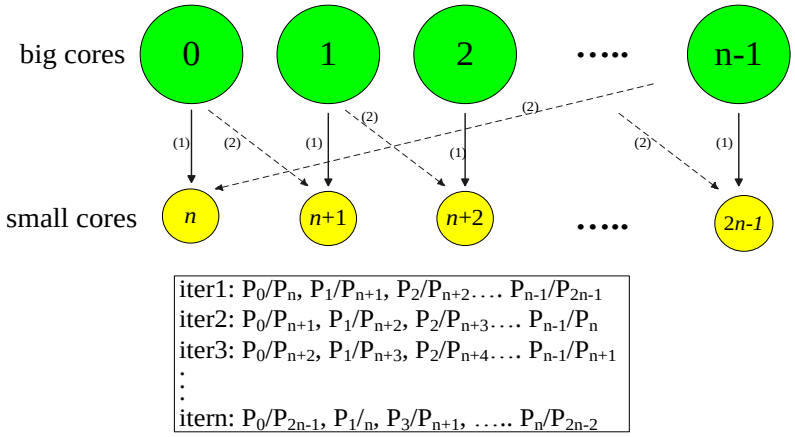


Figure 3-4. Pair-wise comparison illustration for 2n-program scheduling on an *n*B+*n*S platform

For the first two cases, it will generate an ideal situation that a "1" job will be assigned on a big core. For the third case, a "1" job will also be allocated on a big core, no matter which "1" job is selected. Similarly, a "0" job will be set on a big core for the fourth case. However, there

must be a “1” job running on a small core at this point, considering that the number of “1” jobs is equal to the total number of big cores. This implies an opportunity for this “1” job running on a small core to be compared with a “0” job executed on a big core in a future iteration, since we have *n* iterations of parallel comparisons. Thus, any case (4) comparison will fall into case (2) comparison eventually. Based on this analysis, we conclude that all “1” jobs will finally go to big cores, meaning that the optimal schedule is achieved after *n* iterations.

Our algorithm can be further generalized to guide the scheduling on a heterogeneous CMP with non-equivalent number of big and small cores. Let us assume there are *m* big cores and *n* small cores. Therefore, there should be a total of *m* jobs with label “1” and *n* jobs with label “0”. Without loss of generality, we assume that *m* is greater than *n*. In this situation, the PRIM-based approach is capable of reaching the desired scheduling status by performing $\lceil m/n \rceil$ rounds of parallel comparisons described in above as shown in Figure 3-4. In case that *m* is less than *n*, the algorithm is similar but requires $\lceil n/m \rceil$ rounds of parallel comparisons.

Figure 3-5 illustrates the scheduling procedure on such a heterogeneous CMP. As can be noted, the parallel comparisons are conducted within a window whose size is equal to *n* (i.e., the smaller of *m* and *n*). By doing so, we are able to perform *n* iterations of parallel comparisons between *n* big and *n* small cores. Note that the total number of “0” jobs is *n* and total number “1” jobs is *m*. According to the analysis described earlier (where *m* is equal to *n*), after each round of parallel pair comparisons between *n* big cores and *n* small cores, all of the *n* big cores will have “1” jobs running on them. Therefore, after rounds of parallel comparisons, all big cores will have “1” jobs. Meanwhile, all “0” jobs are scheduled running on the small cores.

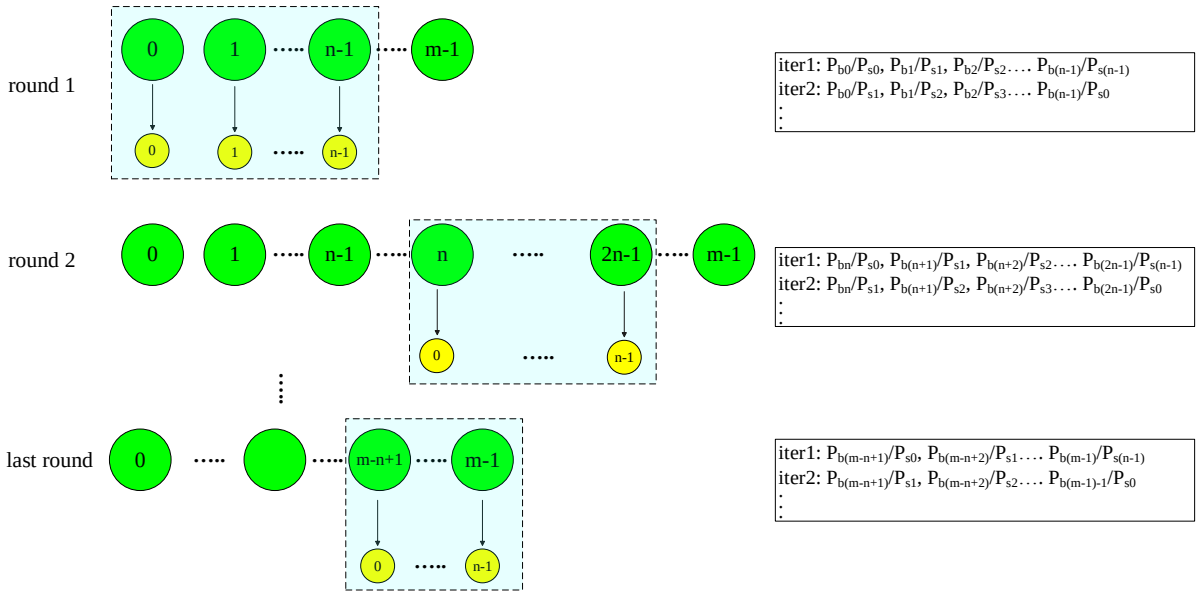


Figure 3-5. Scheduling procedure on a heterogeneous CMP with *m* big cores and *n* small cores (*m* > *n*). Big cores are denoted as Pb*i* (*i*=0,1, …*m*-1) and Small cores are denoted as Ps*j* (*j*=0,1,…*n*-1)

It is important to notice that this approach introduces fairly light overhead to the program execution. First, the model training is conducted offline and therefore has no impact on the dynamic execution. Second and more importantly, the pair-wise comparisons which are performed at each scheduling interval can be completed in reasonable time due to the parallel operation in each round. Specifically, although the total number of comparisons to reach the desired scheduling is approximately $O(mn)$, all comparisons can actually be finished in $O(m)$ time, where m is the larger core count (i.e., $m \geqslant n$ on a CMP with m big cores and n small cores, or the other way around). Note that traditional heterogeneity-aware scheduling policies based on relative performance estimation involve a sorting process in order to identify the programs suitable to run on big cores (or small cores). Assume a quicksort algorithm is employed for the operation. This introduces $O(n\log n)$ comparisons where n is the total number of programs. Therefore, our rule-set based scheduling policy raises no additional overhead compared to state-of-the-art strategies.

Also, the algorithms discussed in this work are built on an assumption that two types of cores are integrated on the die. This is reasonable considering that most commercial heterogeneous chip multi-processors including ARM big.Little [2] and Nvidia Tegra 3 [13] are composed of two families of cores for good tradeoff between the design complexity and energy-efficiency.

3.4 Experimental Setup

3.4.1 Simulation Environment

We use a modified SESC simulator [86] to conduct the experiments in this work. The simulator is configured to contain a number of big and small cores, whose architectural parameters are listed in Table 3-2. McPAT v0.8 [69] is used for dynamic and leakage power estimation. We select 26 programs from SPEC 2000 and SPEC 2006 with the ref input size for the study. In the multi-program simulation, we form 220 workloads composed of individual programs. Note that we do not use other programs from the suites for two reasons: (1) our current cross compiler is only capable of compiling programs implemented with C/C++. Many remaining programs are written in Fortran, thus it is difficult to co-compile them with C/C++ applications; (2) we pay much attention to scheduling-insensitive programs, which are usually not carefully examined in performance-oriented scheduling studies, to demonstrate and exploit the opportunity of energy optimization.

Each program is simulated for 1 billion instructions after fast-forwarding the initial 2 billion. For the single-program study, we use 19 programs for training and use the remaining 7 programs for validation. For the multi-program study, we choose 180 out of the 220 program combinations for PRIM model training and use the remaining ones to evaluate the effectiveness. Recall that the training procedure is conducted offline. This takes about 3 seconds on a Dell Precision T7500 workstation equipped with an Intel E5530 CPU. In addition, for the scheduling in absence of idle

cores, we always launch as many programs as cores. We stop the simulation when the slowest application in the workload completes 1 billion instructions. The faster applications are not repeating. By doing so, we guarantee that the same amount of work is always performed when different scheduling policies are engaged, i.e., each application in the workload executes 1 billion instructions after the initial 2 billion. This makes the comparison of total energy consumption from run to run rational. Note that once faster programs complete, the scheduling problem deprecates to the situation with idle cores.

Table 3-2. Architectural parameters of system components

Component	Parameter	Value
Big core	Pipeline type	out-of-order
	Processor width	4
	ALU/FPU	4/2
	ROB/RF	120/160
	L1I cache size	32KB
	L1D cache size	32KB
	L1 associativity	4
	BTB entries	2048
Small core	Pipeline type	in-order
	Processor width	2
	ALU/FPU	2/1
	L1I cache size	16KB
	L1D cache size	16KB
	L1 associativity	2
	BTB entries	1024
Other parameters	L2 cache size	4MB
	L2 associativity	8
	Cache block size	32B
	Branch Predictor	Hybrid
	Frequency	3G

The scheduling interval is set to 2.5ms in this study. As shown in prior works [33], this granularity is small enough to capture the variations in program execution behaviors and assist the scheduler to make energy-efficient assignments more precisely. We do account for the migra-

tion overhead due to architectural state retrieving and set it to 150 μs [72]. The additional energy dissipation due to the migration is also appropriately modeled. For instance, the energy consumed by cache re-warming can be calculated from the corresponding cache access times. The time overhead of the scheduler is ignorable because making a scheduling decision only requires reading the performance counters from the big and small cores and comparing them with the corresponding rules. We compare performance, energy consumption, and ED product resulting from different schedulers to assess the effectiveness. Note that since each workload executes the same amount of instructions under different scheduling policies, comparing the total energy consumption is equivalent to comparing the energy-per-instruction (EPI). We thus use EPI as the metric for interpretation in later sections. Also note that in scenarios of single-threaded executions, scheduling based on EPI is an approximation of the optimal assignments.

3.4.2 Scheduling Algorithms for Comparison

In this subsection, we introduce the scheduling strategies implemented for comparison.

Static scheduling: This is the baseline scheduler implemented for the comparison. The programs are pinned to processor cores and execute till completion. For the single-program investigation, this means two specific approaches: static-big where the program is mapped to the big core; and static-small where the program goes to the small core. For the multi-program evaluation, we run all possible task-to-core mappings and choose the most energy-saving one as the baseline for comparison.

Round-robin (R-R): With this policy, the programs running on the big and small cores are swapped every 5 intervals. The scheduler does not take into account the program difference and runtime execution behaviors, but blindly swaps the jobs at a preset frequency.

Sample-Optimize-Symbios (SOS): The SOS scheduler is originally proposed for the simultaneous multi-threading execution [98]. Many heterogeneous scheduling algorithms presented in prior works also fall into this category [24][63]. With this scheduling policy, the execution proceeds in a pattern consisting of three steps. First, at a scheduling point, the programs are executed on each type of core for an interval. This is called the "sampling phase" since the energy consumption of each assignment is available after this process. Second, the most energy-efficient scheduling is identified, thus this step is termed the "optimization phase". Finally, the execution will experience the "symbios phase" during which all programs are running *N* intervals with the optimal mapping. In this study, *N* is set to 10. Note that this strategy is also called "sampling" in a few prior works.

MLP-ratio: This scheme is introduced in a recent work [33] aiming to improve the system throughput on heterogeneous CMPs. Although it does not focus on energy saving, it stands as one of the best heterogeneity-aware schedulers to date, thus deserving a comparison with our strategy. Note that the optimal scheduler proposed in [33] takes both the instruction-level parallelism (ILP) and the memory-level parallelism (MLP) into consideration. Nevertheless, the authors demonstrate that the algorithm based on only MLP-ratio delivers fairly close performance to their optimal scheduler. Considering the complexity to calculate the ILP on the fly, we implement a scheduling scheme based on only MLP estimation for the comparison due to its simplicity. In the MLP-ratio scheduler, the memory-level parallelism (MLP) ratios of all programs between the big and small cores are evaluated. Programs with higher MLP ratios are placed on the big cores while those with lower ratios are assigned to small cores.

PRIM: At a scheduling point, the performance metrics collected from a pair of big and small cores are compared with the selective PRIM rules. If the conditions for both big and small

cores are satisfied, the jobs on two cores are swapped; otherwise the current assignment is maintained. In case where the number of cores (programs) is greater than or equal to four, the optimal scheduling is achieved through a few steps of suboptimal assignments as described in section 3.3.

Hierarchical PRIM (H-PRIM): Instead of training a single PRIM model, we use CART to partition the training data into 4 categories according to the performance measurements and train a PRIM model for each subset. At a scheduling point, we first identify to which subset a pair of program executions belongs. We then compare the corresponding PRIM rules with the execution behaviors of these two programs and schedule accordingly.

Oracle: In this scheduling policy, we assume that the scheduler knows the energy consumption of each program mapping in advance and performs the optimal scheduling based on that information. To implement this algorithm, we measure the total energy of each program assignment for each scheduling interval. We then choose the most energy-efficient schedule for the next interval.

3.5 Result Analysis

In this section, we perform a detailed evaluation of the rule-set guided scheduling algorithm by comparing it with a set of existing schemes. We first demonstrate the extracted rule sets used for scheduling and then compare the effectiveness of different schedulers from both energy saving and performance improvement perspective.

3.5.1 Results in Presence of Idle Cores

We start the result demonstration by analyzing the extracted rules. By training PRIM models, we generate two sets of rules respectively for the big core and small core.

Rule set:

Big Core Rules:

L1D.nMiss > 37510 &&

L1D.writeHit < 191275 &&

nStall.SmallReg > 213400

Small Core Rules:

L1D.nMiss < 45600 &&

L2.nAccess > 32200 &&

BR.misp < 27733

As we mentioned in previous section, a matching between the observed execution behaviors and the corresponding rule sets implies that transferring the job to a different type of core is more energy-saving. Specifically, if the program is currently running on a big core and we observe that its cache access and pipeline stall statistics satisfy the big core conditions listed above, it should be moved to a small core for the execution in next interval. The two inequalities related to L1 data cache (*L1D.nAccess* and *L1D.writeHit*) indicates that the execution in the past interval issues considerable memory requests that go to the L1D, however, many accesses are missed in this level of cache. The third condition shows that the pipeline is frequently stalled due to the shortage of free physical registers (*nStall.SmallReg*). Jointly, these three conditions indicate that the program may not be able to effectively utilize the computation resource on the big core and would be more suitable to run on a small core for better energy efficiency. Note that all the coun-

ter values are normalized to those in one million instructions (e.g., *L1D.nMiss* is actually *L1D.nMiss*/*MInst*). On the other hand, the rules corresponding to the small core imply that the program can achieve high speedup on the big core and result in better energy efficiency after migration. For instance, the relatively low miss rate in the L1 data cache and infrequent branch mispredictions means that the program is able to fully exploit the computing resource on the big core and more efficiently utilize the energy (i.e., executing with a lower EPI).

We now compare the effectiveness of different scheduling policies on reducing the energy consumption. Figure 3-6 demonstrates the comparison of energy for all selected programs running on a dual-core heterogeneous CMP when different strategies are engaged. Note that the results under all schemes are normalized to that corresponding to the big core execution. As can be observed, the selected programs manifest distinctive variation on the energy consumption. For the static schemes, applications including *equake*, *lbm*, *mcf*, and *milc* are more appropriate to run on the small core while benchmarks such as *wupwise*, *dealII* and *h264* are suitable candidates to be placed on the big core. This corroborates the conclusions drawn by few prior works that program features such as memory-intensity, computation-level and memory-level parallelism impact their relatively energy consumption on different types of cores [24][63][100], which further justifies the opportunities for intelligent scheduling on heterogeneous systems.

The round-robin scheme does not involve true scheduling intelligence either, since it just blindly transfers the program to a different core at a preset frequency. As a consequence, it coincidently results in lower energy consumption than the static scheme for some benchmarks while performing even worse for programs including crafty and eon. The SOS scheme is able to identify the correct task-to-core mapping via online sampling, thus leading to lower energy consumption than both static and R-R for many benchmarks. However, it suffers from two intrinsic draw-

backs. First, frequent sampling introduces noticeable overhead which may prolong the execution time and consume extra energy that mitigates the benefit. Second, this scheduler assumes a continuum of program characteristics in the symbios stage (i.e., the following *N* intervals after sampling), which might not be true as the execution behaviors usually vary across different phases. This may cause inefficient executions in many intervals and thus raise the energy consumption. The PRIM rule set guided policy works the most closely to the oracle scheduler because it eliminates the unnecessary sampling overhead and transfers the job to the energy-saving core when necessary. In general, the round-robin, SOS, PRIM and oracle schedulers are able to reduce the energy consumption respectively by 3.4%, 12.8%, 20.1% and 22.7% compared to the execution on big core.

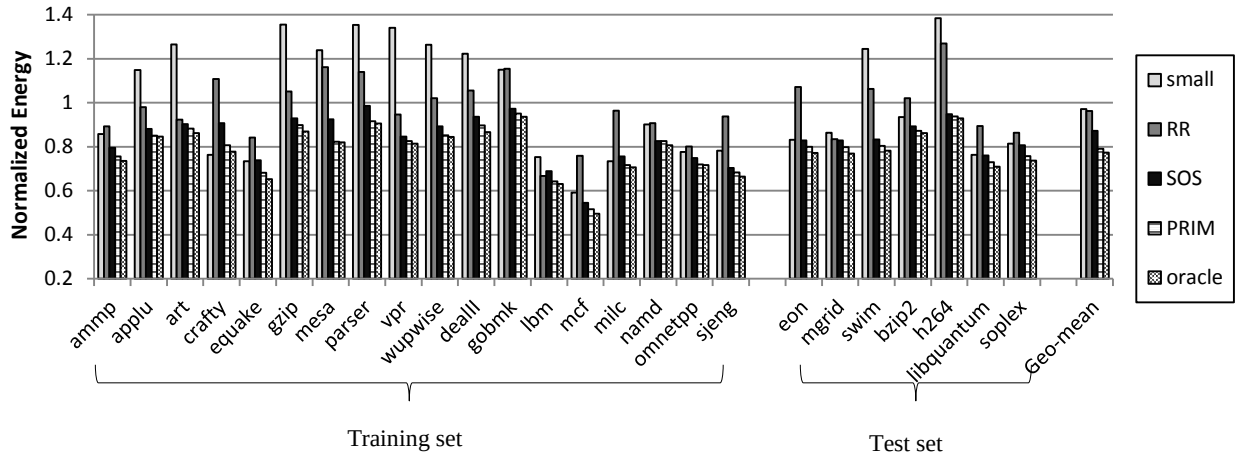


Figure 3-6. Normalized energy consumption for single-programs executing on a dual-core CMP with different schedulers

The unnecessary context switch is an important cause of inefficient execution on heterogeneous CMPs when non-ideal schedulers are employed. Those context switches introduce substantial overhead due to architectural state retrieving and cache re-warming to the execution; furthermore, they transfer programs to inappropriate cores for execution, which may adversely increase the energy consumption. Taking this into consideration, we collect the number of context

switches during the execution for each application when different schedulers are used. Table 3-3 lists the recorded statistics. The round-robin scheme causes many more switches than all other policies since it continually moves a job every 5 intervals (recall the experimental set up described in section 3.4). The SOS scheme makes a job migration decision based on the sampling result and thus usually moves jobs at a much lower frequency, which in turn significantly reduces the switch times.

Table 3-3. Number of context switches

Benchmark		RR	SOS	PRIM	Oracle
Training	ammp	1084	201	194	189
	applu	486	110	127	140
	art	466	107	89	83
	crafty	384	98	93	101
	equake	874	162	177	182
	gzip	658	149	106	96
	mesa	336	58	69	71
	parser	516	193	102	123
	vpr	464	122	97	103
	wupwise	298	44	58	72
	dealII	354	102	118	105
	gobmk	464	121	102	99
	lbm	734	101	121	115
	mcf	1268	406	387	367
	milc	1208	104	97	89
	namd	358	76	64	58
	omnetpp	382	152	125	104
	sjeng	496	101	95	98
Test	eon	360	89	103	99
	mgrid	959	120	98	88
	swim	980	103	105	95
	bzip2	1072	82	105	102
	h264	304	98	75	72
	libquantum	322	60	73	70
	soplex	496	93	120	114

The PRIM and the oracle scheduler generally lead to comparable context switches as SOS does; however, they capture the migration opportunities more precisely and make scheduling de-

cisions at finer-granularity, thus appearing to be more energy-saving compared to SOS. Also, since the PRIM scheduler does not require sampling, it offers better performance than the SOS scheme. Figure 3-7 shows the average performance of each program normalized to the big core's execution. Not surprisingly, the PRIM scheduler results in only 20% longer execution time than the static-big policy, delivering better performance than both RR and SOS which respectively prolong the running time by 69% and 29%.

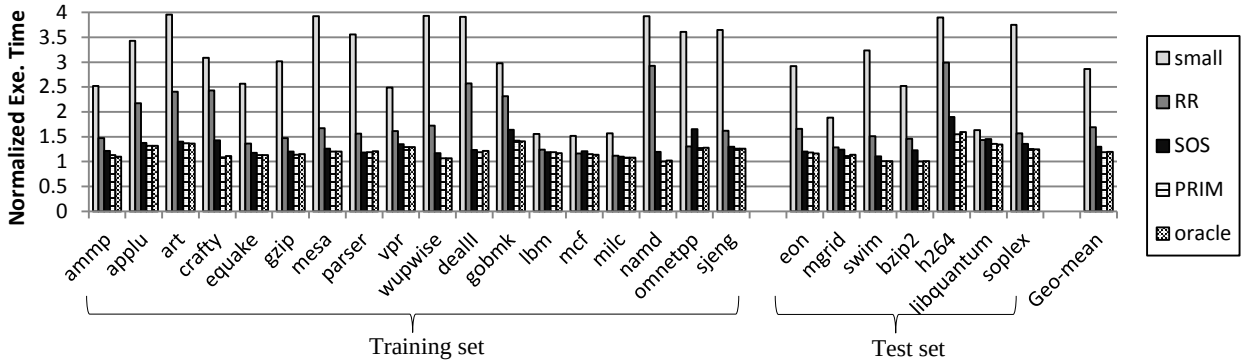


Figure 3-7. Normalized execution time for single-programs executing on a dual-core CMP with different schedulers

3.5.2 Results without Idle Cores

In this subsection, we demonstrate the evaluation results for the second circumstance, where each workload contains as many programs as cores. Again, we analyze the extracted PRIM rules at first. By training a PRIM model, we can generate the following rule sets to guide the scheduling for two programs running on a pair of big and small cores.

Rule set:

Big Core Rules:

L1D.nMiss < 13430 &&

nStall.SmallIQ > 256949

Small Core Rules:

BP.nMiss > 9674 &&

iLoad.count > 198169 &&

iALU.count < 400026 &&

L1D.nMiss < 17701

Note that at a scheduling point, the scheduler compares the performance metrics collected from a pair of big and small cores. If the measurements on both sides are satisfied with these rules, the scheduler predicts that swapping the two programs will decrease the total energy consumption. This is different from the operation described in the previous subsection, where the execution behaviors from either the big core or the small core are compared against the corresponding rules. Since our prediction is made at each scheduling interval, the goal essentially translates to lowering down the total power of that period. Also, the big cores always consume much larger power than the small ones, thus dominating the total power consumption all the time. We present these two statements to assist the interpretation of the PRIM rules. Note that we will analyze the correlation between execution behaviors and power consumption including both dynamic power and leakage power.

Let us focus on the big core rules at first. As can be seen, the big core rules suggest that a program with a low L1 cache miss rate (*L1D.nMiss*) and substantial internal stalls (e.g., stalls due to small instruction issue queue, or *nStall.smallIQ*) should be exchanged to the small core for execution. It is straightforward to understand the metrics related to L1 cache and instruction

fetch. A Low L1 cache miss rate indicates that the current program on the big core is executed at relatively high speed without suffering from frequent cache misses. However, from the power perspective, this implies large dynamic power on many function units due to high activities. As for the second condition, a large *nStall.smallIQ* value indicates that the program spends substantial time on waiting for free IQ entries, meaning that the IQ is always full during this interval. This eventually leads to high utilization in the IQ and increased dynamic power consumption on this component because of frequent operations such as checking the operands' status. On the other hand, components including IQ and integer ALU tend to become the hotspot on die. As a consequence, the leakage power on these units is rapidly increasing since it is proportional to the temperature. In one word, the big core rules outline the features of intervals which tend to consume both high dynamic and leakage power. For the purpose of energy saving, these intervals should be migrated to the small core for execution.

The small cores rules work in tandem with the conditions on big cores. Recall that the total power consumption is dominated by the big core. Therefore, the rules for the small core essentially characterize the execution phases that are not probable to result in extreme high power on a big core. Specifically, the first and the third conditions respectively set constraints for the occurrences of branch mispredictions (*BP.nMiss*) and number of integer ALU instructions (*iALU.count*). A branch misprediction will lead to a pipeline flush and lower down the execution speed. Fewer number of ALU instructions can alleviate the utilization on ALUs, reducing the dynamic power and cooling down the component accordingly. These two conditions jointly reduce the power consumed by the core running this program. On the other hand, the rule sets require that the amount of load instructions (*iLoad.count*) should be no smaller than a certain value while the misses in L1 data cache (*L1D.nMiss*) cannot go beyond a threshold. These two condi-

tions imply that this program potentially issues a large amount of memory requests, but most of them can be served by the L1 cache. Nevertheless, the stress on L1 cache will not significantly increase the total power consumption since the L1 cache consumes relatively small power compared to other components. In general, the intervals filtered by the small core rule set tend to result in moderate power if executed on the big core, thus reducing the chip-level power consumption.

As described in section 3.2.2, CART is able to partition the entire data set into several subsets, each of which contains similar samples. Therefore, if we train an individual PRIM model for each subset, the effectiveness of our strategy is expected to be increased due to the similarity of instances within the same subset. Taking this into account, we use CART to perform a data segmentation operation prior to the PRIM model training. Figure 3-8 demonstrates the segmentation result for all training instances. As can be seen, the entire data set are partitioned into 4 categories, as represented by the 4 leaf nodes on the generated tree. Each branch represents a condition on the performance metrics on a big or small core and is expressed in a form of "$X_i \geqslant$ (or $\leqslant$) M", where X_i denotes a performance metric and M denotes a value to segment the data set. Specifically in the tree shown in Figure 3-8, *X126* indicates the number of branch mispredictions on the big core (*BP.nMiss*) and *X232* corresponds to the *iALU.count* metric which records the number of integer ALU instructions on the small core. *X244* tracks the number of fetched instruction on the small core (*nFetch*). The value at each leaf node is the average of CART response for that partition.

We train a PRIM model for each data segment and list their respective rule sets in Table 3-4. Note that for the fourth subset (i.e., the rightmost leaf node in Figure 3-8), more than 95% of the

included samples maintain a flag “1”, meaning that the majority of this partition are candidates for job swap. Consequently, we do not train an extra PRIM model for this subset and directly use its branch conditions to guide the scheduling. During the execution, this tree is accessed at each scheduling point in order to classify a program pair into an appropriate subset. The access starts from the root node of the tree. If the condition is satisfied after the variable comparison (*X126*, or *BP.nMiss*), the access will proceed to the left child; otherwise it goes to the right child. This process is performed again on the child node and the program pair will be classified to a specific subset thereafter. The corresponding PRIM rules are then compared with the execution behaviors and make job assignments accordingly. In particular, if a program pair falls into the rightmost subset, the scheduler will immediately swap the two jobs for execution in the next interval.

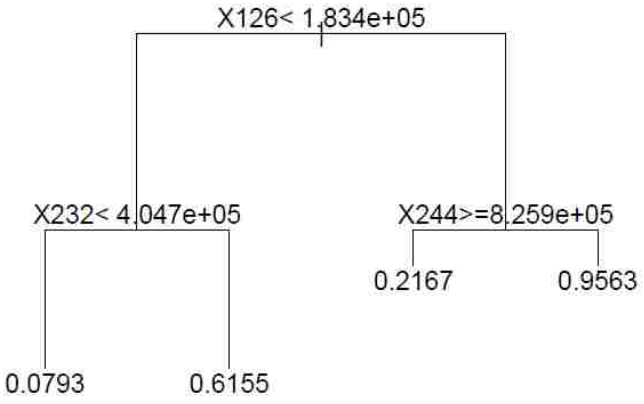


Figure 3-8. Data segmentation result from CART

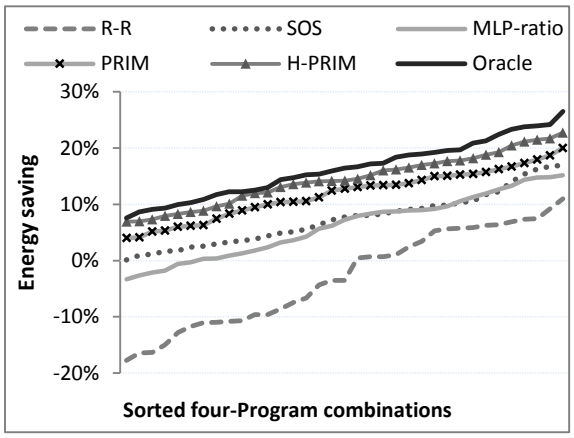
We now compare the effectiveness of different scheduling policies on reducing the energy consumption and improving energy efficiency. To demonstrate the scalability of our proposed algorithm, we run four-program workloads on two types of heterogeneous architectures: a platform with an identical number of big and small cores (2B+2S) and a system with non-equivalent numbers of big and small cores (3B+1S).

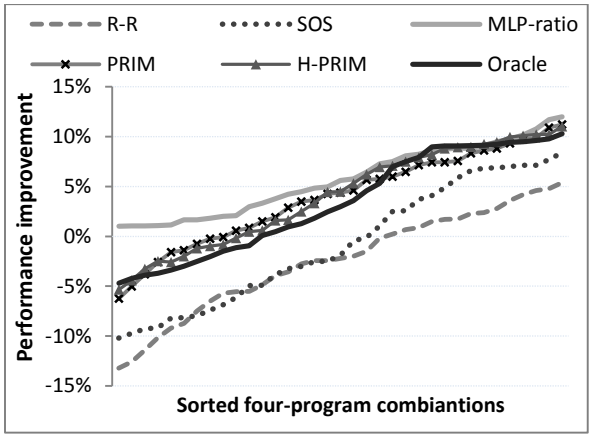
Table 3-4. PRIM rule set for each data segment

Segment	Big Core Rules	Small Core Rules	Performance metrics description
Subset 1	*LDSTUnit.util* < 0.27 && *nStall.noCachePort* < *10785*	*L1I.nMiss* > *3791*	*LDSTUnit.util*: the utilization of load/store unit *nStall.noCachePort*: cumulative stall cycles due to cache port contention *L1I.nMiss*: number of misses in L1I cache
Subset 2	*IQ.avgFree* < 3 && *L1I.nHit* > 817392	*avgBranchPenalty* > *73* && *iComplex.count* < *3921*	*IQ.avgFree*: the average number of free entries in IQ, indicating the IQ utilization *L1I.nHit*: number of hits in L1I cache *avgBranchPenalty*: average penalty (in cycles) of branch misprediction *iComplex.count*: number of integer complex instructions, such as division
Subset 3	*nStall.noCachePort* < *44750* && *nStall.smallREG* > *851493*	*L1I.avgMissLat* < *513*	*nStall.smallREG*: cumulative stall cycles due to available registers *L1I.avgMissLat*: average penalty (in cycles) of a miss in L1I cache
Subset 4	N/A		

Figure 3-9 (a) illustrates the energy reduction for these workloads on the first platform (2B+2S) when distinct schedulers are engaged. Note that all results are compared with those corresponding to the static scheduling case. The workloads are sorted in ascending order according to the degree of energy saving. As can be observed, our PRIM and H-PRIM strategies always outperform other scheduling algorithms with respect to energy consumption. This is because the rule-set guided scheduler is capable of effectively identifying the most appropriate program assignment at runtime to minimize the energy consumption. Furthermore, the total energy consumed by H-PRIM is fairly close to the oracle case (i.e., the minimum), since the data segmentation increases the accuracy of identifying the candidate intervals. Other schedulers suffer from distinctive drawbacks which adversely impact their effectiveness. The previous subsection explains the disadvantage of the R-R and SOS scheduler. The MLP-ratio algorithm, on the other hand, aims to improve the system performance. As we demonstrated in section 3.1, this scheduler can increase the total energy consumption for some intervals, thus trailing our strategies in saving the total energy.

Figure 3-9 (b) shows the performance improvement when the workloads are respectively running with these schedulers. Note that the workloads are sorted according to the performance gain in this figure. Also recall that the oracle scheduler is the optimal with respect to the energy consumption instead of performance. As can be seen, the MLP-ratio strategy always leads to better performance (i.e., positive value) compared to the baseline. This does not go beyond our expectation because the goal of this scheduler is to enhance the overall performance, thus the programs are assigned in a manner to maximize the execution speed. On the other hand, both performance improvement and degradation (i.e., negative value) are observed in other scheduling policies. The performance loss mainly stems from two sources, namely migration overhead and slower execution in certain intervals. Our scheduler eliminates unnecessary job swaps during the execution compared to round-robin and SOS, thus delivering better performance.





(a) Energy saving (b) Performance improvement

Figure 3-9. Evaluation results of four-program workloads running on a 2B+2S platform

Figure 3-10 demonstrates the average performance gain, energy saving and ED reduction for all workloads. From the performance respective, MLP-ratio stands as the optimal by accelerating the execution by 5.7% while PRIM and H-PRIM respectively enhance the performance by 3.9%

and 4.1%. Note that the MLP-ratio scheduler is more effective for scheduling-sensitive work-loads [33]. However, the performance gains for applications demonstrating less sensitivity to program assignment are fairly modest. Therefore in general, our schedulers lead to comparable performance to MLP-ratio on average. For energy saving, the PRIM and H-PRIM algorithms are able to reduce the energy consumption by 11.8% and 14.8% compared to 16.3% delivered by the oracle scheduler. Finally, for the ED metric, the PRIM and H-PRIM algorithms respectively reduce its value by 15.3% and 17.9% while the oracle scheduler can decrease the product by 19.1%. The SOS and MLP-ratio policies lead to less impressive savings. In other words, our best scheduler H-PRIM outperforms the MLP-ratio policy, which is one of the optimal state-of-the-art heterogeneous schedulers by 7.8% and 5.7%, respectively, on energy and ED. We also collect the number of context switches with different scheduling policies and observe a similar trend as shown in Table 3-3.

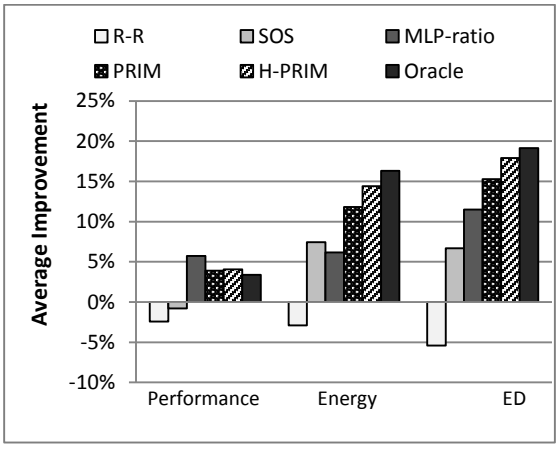
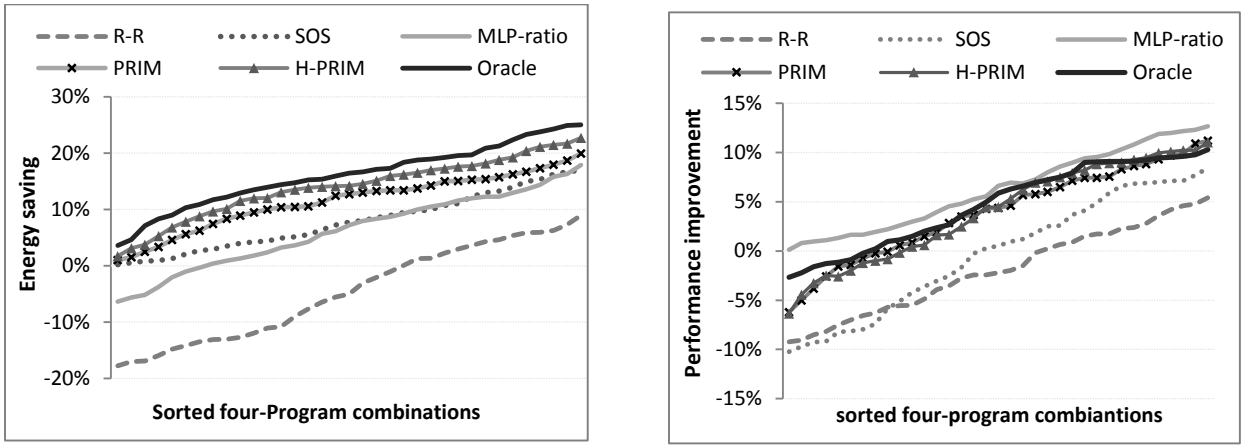


Figure 3-10. Average improvement in energy, performance and ED of all four-program work-loads running on a 2B+2S platform

Figure 3-11 and Figure 3-12 demonstrate the results for quad-program work-loads running on a CMP consisting of three big cores and a small core (3B+1S). As can be seen, the general trends of the curves are similar to those shown in the 2B+2S case. On average, the PRIM and H-

PRIM scheduler is capable of saving energy by 11.5% and 13.9%. For the energy-delay product, these two schemes decrease the value by 14.9% and 17.3%. This implies that compared to MLP-ratio, the H-PRIM policy reduces the total energy and ED by 8.1% and 5.5%, respectively.



(a) Energy saving (b) Performance improvement

Figure 3-11. Evaluation results of four-program workloads running on a 3B+1S platform

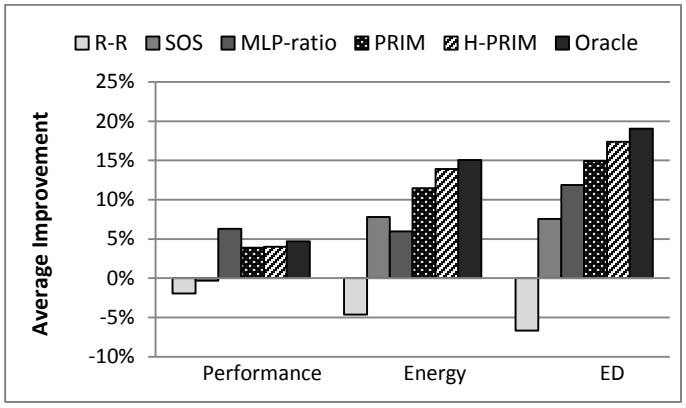


Figure 3-12. Average improvement in energy, performance and ED of all four-program workloads running on a 3B+1S platform

We also evaluate the effectiveness of our strategy with other configurations. Figure 3-13 shows the average improvement when two-program workloads are running on a CMP with a big and a small core. Not-surprisingly, H-PRIM surpasses other policies by improving the energy-

efficiency most closely to the oracle scheduler. Nevertheless, our evaluation results demonstrate that the proposed rule-set guided scheduling policy is more effective in optimizing the energy-efficiency of single-ISA heterogeneous platforms compared to existing schedulers.

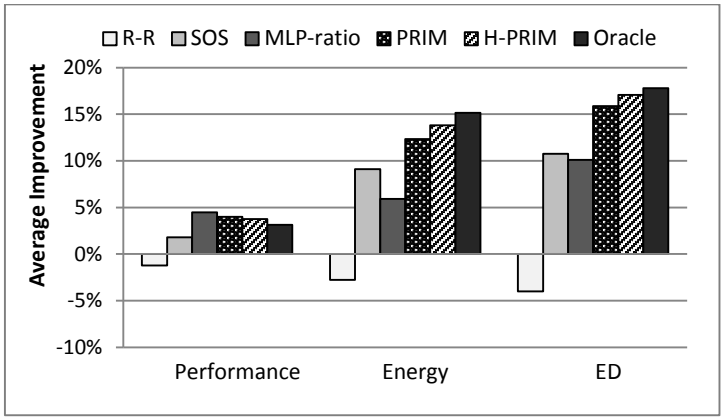


Figure 3-13. Average improvement for performance, energy and ED for two-program workloads running on a 1B+1S platform

3.6 Related Work

Within past years, several researchers have authored outstanding studies in the heterogeneous architecture field. Kumar et al. [63] propose one of the earliest single-ISA heterogeneous multiprocessors and discuss its potential for power reduction. The sampling-based scheduling algorithm that can be applied to a realistic multiprocessor for energy-efficient execution is also proposed. In [65], the performance for multithreaded workload executing on a single-ISA heterogeneous processor is analyzed in detail. By adopting a similar sampling-based assignment policy, the system can capture the intra-thread diversity and schedule the jobs for the maximal throughput. Becchi and Crowley evaluate a set of static and dynamic scheduling policies designed for heterogeneous platform in [24]. The authors show that dynamic job scheduling largely outperforms the static assignment by delivering higher throughput. Hao et al. [49] describe a scheduling policy using hardware counters and evaluate it on a real multiprocessor system run-

ning Linux. They argue that the last level cache access latency is a good metric to guide the scheduling on heterogeneous platform. In [61], Koufaty et al. introduce the bios scheduling which is similar to the policies based on memory intensity. Balakrishnan et al. quantitatively analyze the impact of performance asymmetry between cores on the application scalability and predictability [22].

Saez et al. present a series of works that target the performance enhancement on asymmetric CMP platforms [88][89][90]. They propose an algorithm named HASS [90] to guide the job assignment on single-ISA heterogeneous systems for the maximum performance. They also develop the CAMP scheduler to explore both efficiency and TLP [88][89]. Radojkovic et al. [24] consider a scenario with massive multithreaded processors where an exhaustive search for the optimal task assignment is unfeasible due to the substantial possibilities. Therefore, they introduce a statistical approach to seek the best work distribution. Li et al. [70] implement a scheduler composed of fast-core-first assignment and migration on a performance-asymmetric CMP architecture. More recently, Craeynest et al. present a heterogeneous scheduler via performance impact estimation (PIE) [33]. Their evaluation results demonstrate that the PIE scheduling policy outperforms prior schemes based on program memory intensity. The authors also show that memory-level parallelism ratios of programs provide good estimation for relative performance and can be employed to guide the runtime scheduling. A similar strategy through the prediction of CPI across core types is proposed by Srinivasan et al. [100].

Studies addressing energy minimization on heterogeneous platform can also be found in literature. Saad et al. [87] and Goraczko et al. [43] respectively propose the software partitioning approach to reduce the energy consumption on heterogeneous embedded systems. In [30], Chen and John present a scheduler based on weighted Euclidean distances to improve the energy effi-

ciency on heterogeneous CMPs. Sharifi et al. [94] takes temperature into account and introduce a joint solution for thermal and energy management. Grant et al. [45] introduce a scheduling mechanism to save energy on asymmetric multiprocessors for scientific applications. In their proposed algorithm, one core is reserved for running the operating system at adjustable frequencies while other processors are executing the user threads at full speed. In [51], Heath et al. design a heterogeneous server cluster which demonstrates remarkable energy efficiency improvement over traditional homogeneous clusters. Singh et al. [96] propose a prediction based approach for power estimation and scheduling on traditional homogeneous CMPs, in order to improve the energy efficiency.

3.7 Conclusion

In this chapter, we propose a scheduling strategy for energy-efficient execution on single-ISA heterogeneous chip-multiprocessors. We demonstrate that performance-oriented scheduling may lead to executions that are not sufficiently energy-efficient. Due to this limitation, we concentrate on energy saving and introduce a rule-set guided scheduling to exploit the optimal energy efficiency on heterogeneous CMPs. We employ advanced statistical tools including PRIM and CART to facilitate the development of our algorithm. The evaluation results show that our proposed algorithm impressively outperforms existing scheduling schemes by minimizing the energy consumption, thus delivering better energy efficiency.

CHAPTER 4. MITIGATING NBTI DEGRADATION ON GPUS THROUGH EXPLOITING DEVICE HETEROGENEITY

4.1 Motivation and Overview

As we shift into the deep submicron era, innovative materials and device architectures are becoming ever demanding to continue the trend toward smaller and faster transistors. Among all candidates in investigation, the Fin field-effect-transistor (FinFET) stands as one of the most promising substitutes for traditional devices at the ensuing technology nodes, since it presents several key advantages over its planar counterpart [7][21][55][59]. By wrapping the conducting channel with a thin vertical "fin" which forms the body of the device, the gate is coupled tighter with the channel, increasing the surface area of the gate-channel interface and allowing much stronger control over the conducting channel [7]. This effectively relieves the so-called short channel effects (SCE) that are observed on planar transistors manufactured with sub-32nm technology, which in turn implies that FinFET devices can provide superior scalability in the deep submicron regime [7].

Another cornerstone motivating the realization of FinFET is the potential performance gain. FinFET transistors can be designed with lower threshold voltage (V_t) and operate with higher drive current, leading to faster switching speed compared to conventional planar devices [1]. Released documents from industry demonstrate that the FinFET transistor persistently demonstrates shorter delay than the planar one while the support voltage is varying, enabling the design and manufacturing of faster processors. Public documents from leading manufacturers also show that the FinFET structure is capable of largely decreasing leakage when the transistor is off [7]. Recently, the Ivy Bridge [3] and Haswell central processing units [4] released by Intel have com-

mercialized this structure (i.e., referred to as "Tri-gate transistor" by Intel), which is also expected to be adopted by other semiconductor manufacturers on their upcoming products [15].

Nonetheless, FinFET is not an impeccable replacement of traditional devices as it raises many challenges to the current industry. One of the most daunting conundrums is the increasing aging rate caused by negative bias temperature instability (NBTI). Recent experimental studies demonstrate that FinFET transistors are more vulnerable to NBTI, leading to a shorter lifetime than a planar device [46][112]. The NBTI aging rate is evaluated by the increase of delay on the critical path after a certain amount of service time. A chip is considered as failed when the delay increment exceeds a pre-defined value after which the timing logic of the processor cannot function correctly. Based on well-established models, we observe that under the same operation condition, the FinFET device degrades much faster than the planar counterpart, implying a significantly reduced service lifespan of the target processor. This clearly spurs the development of new techniques to circumvent this problem and prolong the lifetime of FinFET-made processors.

Fortunately, the brief comparison between planar and FinFET transistors sheds some light on alleviating the NBTI effect on future processors. By effectively exploiting the device heterogeneity and leveraging the higher NBTI immunity of planar transistors, the aging of the FinFET structures can be largely suppressed. In this chapter, we propose a set of techniques built on top of this principle to improve the durability of FinFET processors. In general, our techniques are implemented by replacing an existing structure with a planar-device equivalent. Along with minor modifications at the architectural level, our proposed techniques are essentially transferring the "aging stress" from the vulnerable FinFET components to the more NBTI-tolerable planar structures, which in turn reduce the accesses to the hardware in study, lower down its activity and temperature, and thus considerably mitigate the NBTI degradation. Note that these strategies

are practically feasible because of the good compatibility between the FinFET and planar process technology [20][32][36].

Considering that the general-purpose graphics processing unit is becoming an increasingly important block in a wide spectrum of computing platforms, we choose a modern GPU as the target architecture to evaluate the effectiveness of our proposed strategies. In this chapter, we mainly concentrate on optimizing the reliability of memory-like structures in the GPU. However, the techniques described in this work can be simply applied to CPU for NBTI mitigation as well. In general, the main contributions of this work are as follows.

- To the best of our knowledge, this work is the first attempt to address the NBTI alleviation at the architectural level for future GPUs manufactured with FinFET.
- We propose a hybrid-device warp scheduler for reliable operation. By decoupling the warp scheduling into two steps of operations and conducting the prerequisites evaluation in a planar-device structure, we eliminate a large amount of read accesses to the FinFET scheduler hardware and considerably alleviate the NBTI effect.
- We develop a hybrid-device sequential-access cache architecture. All memory requests to this cache are handled in a serialized fashion such that the tag-array made of planar transistors is probed first and the matching block in the FinFET data array is accessed on a cache hit. This reduces the activity on the cache data array and improves its reliability.

4.2 Background

4.2.1 NBTI Degradation Mechanism

Negative Bias Temperature Instability is becoming one of the dominant reliability concerns for nanoscale P-MOSFETs. As explained by the classical Reaction-Diffusion model [75], NBTI

is caused by the interaction of silicon-hydrogen (Si-H) and the inversion charge at the Si/oxide interface [18]. When a negative voltage is applied at the gate of PMOS transistors, the Si-H bonds are progressively dissociated and H atoms diffuse into the gate oxide. This process eventually breaks the interface between the gate oxide and the conducting channel, leaving positive traps behind. As a consequence, the threshold voltage of the PMOS transistor is increased, which in turn elongates the switching delay of the device through the alpha power law [91]:

$$T_s \propto \frac{V_{dd}L_{eff}}{\mu(V_{dd}-V_t)^{\alpha}} \quad \text{4.1}$$

where μ is the mobility of carriers, α is the velocity saturation index and approximates to 1.3. L_{eff} denotes the channel length. The process described above is termed the "stress" phase where the threshold voltage is increasing with the service time, modeled by the following equation [111].

$$\Delta V_{tstress} = \left(\frac{qT_{ox}}{E_{ox}}\right)^{1.5}.K.\sqrt{C_{ox}\left(V_{gs}-V_t\right)}.e^{\frac{-E_a}{4kT}+\frac{2(V_{gs}-V_t)}{T_{ox}E_{01}}}.T_0^{-0.25}.T_{stress} \quad \text{4.2}$$

However, when the stress voltage is removed from the gate, H atoms in the traps can diffuse back to the interface and repair the broken bond. This results in a decrease in the threshold voltage, thus termed the "recovery" stage. As illustrated in Figure 4-1, the iterative stress-recovery processes lead to a saw-tooth variation of the threshold voltage throughout the device's lifespan. The final V_t increase taking both stress and recovery into account can be computed as:

$$\Delta V_t = \Delta V_{tstress}.\left(1-\frac{2\xi_1 T_{ox}+\sqrt{\xi_2 e^{\frac{-E_a}{kT}}T_0 T_{stress}}}{(1+\delta)T_{ox}+\sqrt{e^{\frac{-E_a}{kT}}\left(T_{stress}+T_{recovery}\right)}}\right) \quad \text{4.3}$$

Note that in equations 4.2 and 4.3, T_{stress} and $T_{recovery}$ respectively denote the time under stress and recovery. Other parameters are either constants or material-dependent variables and are listed in Section 4.4.

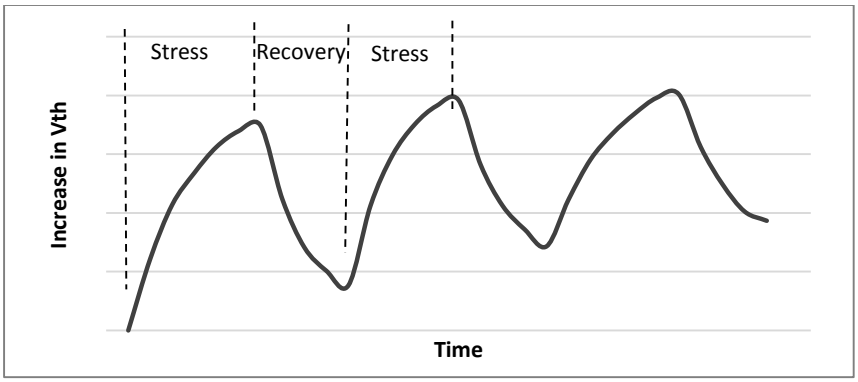
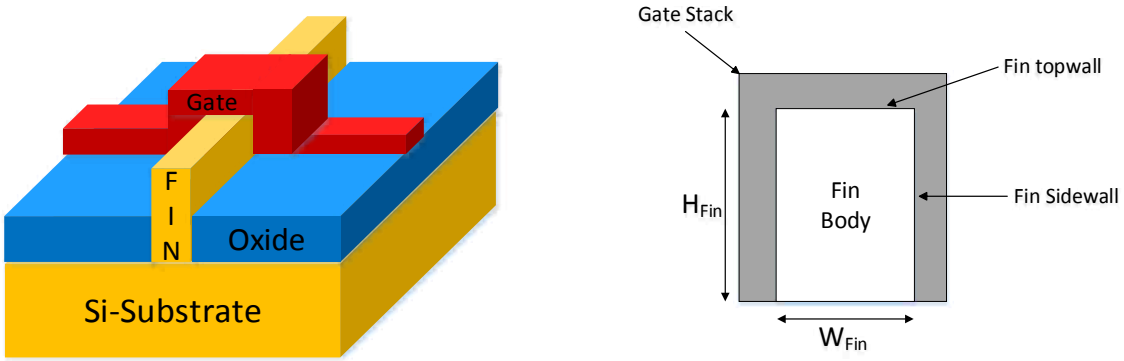


Figure 4-1. NBTI degradation containing stress and recovery phases

That FinFET devices are more vulnerable to NBTI is generally attributed to their unique non-planar architecture, which is visualized by Figure 4-2. As can be seen, compared to a traditional planar transistor, the FinFET structure is designed with additional fin sidewall surface with higher availability of Si-H bonds [46][112], implying larger chances of forming interface traps and consequently expediting the device degradation.



(a) Overview (b) Side view

Figure 4-2. FinFET transistor structure

The NBTI aging rate depends on multiple factors including both circuit parameters and workload execution patterns. In general, it is acknowledged that voltage, temperature, and the stress/recovery time have strong impact on the aging rate [18][106]. In this work, our proposed

techniques significantly reduce the accesses to the target structures, thus lowering the localized activity and temperature, which is beneficial in enhancing the structure durability.

4.2.2 Target GPU Architecture

The prevalence of unified programming languages (e.g., CUDA, OpenCL) has made the general-purpose graphics processing unit a core component in a large variety of systems ranging from personal computers to high-performance computing clusters. Therefore, it is highly important to alleviate the NBTI degradation on this ever increasingly important platform.

Figure 4-3 visualizes the architectural organization of a representative GPU. Note that we follow the Nvidia terminology to depict the processor architecture. As can be seen, the major component of a modern GPU is an array of Streaming Multiprocessors (SMs), each of which contains a number of CUDA cores (SPs), load/store units and special function units (SFUs). A CUDA core is responsible for performing integer ALU and floating point operations while the SFUs are devoted to conducting transcendental operations such as sine, cosine, and square root. Each stream multiprocessor also contains a register file, a shared memory and a level 1 cache (usually including instruction/data/constant/texture caches) that are shared among all threads assigned to the SM. All stream multiprocessors connect to an interconnection network, which transfers the memory requests/services between the SMs and the shared L2 cache.

An application developed in CUDA (or OpenCL) contains at least one kernel running on the GPU. A typical kernel includes several blocks composed of substantial threads. During a kernel execution, multiple blocks are assigned to an SM according to the resource requirement. A group of threads from the same block form a warp treated as the smallest scheduling unit to be run on the hardware function units in an SIMT fashion.

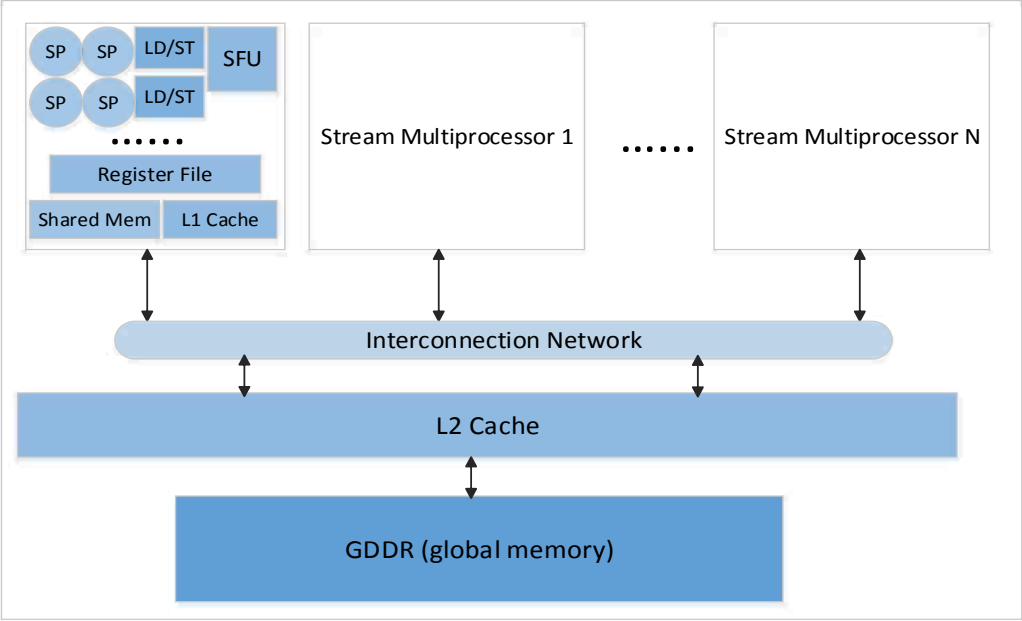


Figure 4-3. An illustration of typical GPGPU architecture

4.3 Mitigating the NBTI Degradation on GPU

As an emerging platform that targets for massively parallel computing domains, a modern GPU is designed with several unique characteristics different from a regular CPU. In this section, we concentrate on the warp scheduler equipped by each SM and the shared L2 cache, in order to investigate the opportunities to slow down the aging on a GPU. By observing representative execution behaviors of a large collection of GPU applications, we propose a set of techniques employing the device heterogeneity to alleviate the NBTI degradation. As we will demonstrate shortly, the proposed techniques do not introduce any additional component to the existing GPU architecture, thus minimizing the hardware cost for the implementation.

4.3.1 Hybrid-device Warp Scheduler

To improve the thread-level parallelism (TLP) and maximize the execution throughput, a modern GPU usually allows multiple warps to reside on the same streaming multi-processor and

hides the execution latencies by switching among those resident warps. At any instant, a warp is considered as ready for execution only when several constraints are simultaneously satisfied.

A first-order prerequisite is the functional correctness, which is secured by ensuring data dependencies between warp instructions. When a warp cannot be dispatched because of unsatisfied data dependency, it should wait until all of its operands are ready. A scoreboard hardware structure is responsible for keeping track of data dependencies in a modern GPU. In addition, warps on a streaming multi-processor contend for limited functional units. When the dispatch port of the functional unit a warp needs to use is not vacant, the warp cannot be issued even when its data dependencies have been satisfied.

The warp scheduler is an SRAM hardware structure in charge of selecting candidates from all resident warps to dispatch. For the purpose of high performance, a warp scheduler is capable of dispatching one warp per clock cycle, requiring that scanning through all the scoreboard entries and querying the dispatch ports of all functional units should be performed at each cycle [52][58]. Figure 4-4 illustrates the high-level organization of a warp scheduler equipped in an SM to elaborate the scheduling process. As shown in the figure, all entries, each of which stores complete information of a warp instruction, are going through the conditions checking in parallel in order to identify the candidates ready for execution. Note that to minimize the delay, the scheduler must read the detailed information of a warp (warp ID, opcode, etc) while evaluating the constraints so that it can dispatch warps as soon as they are ready. Selected warps are sent to the appropriate function units according to the instruction opcode afterwards.

This particular design naturally inspires a technique to mitigate the NBTI degradation on the scheduler. If the readiness of all warp instructions are known ahead via a certain “predicate”,

then only the entries with all constraints met are accessed, which in turn decreases the localized activity and temperature, and improves the structure durability.

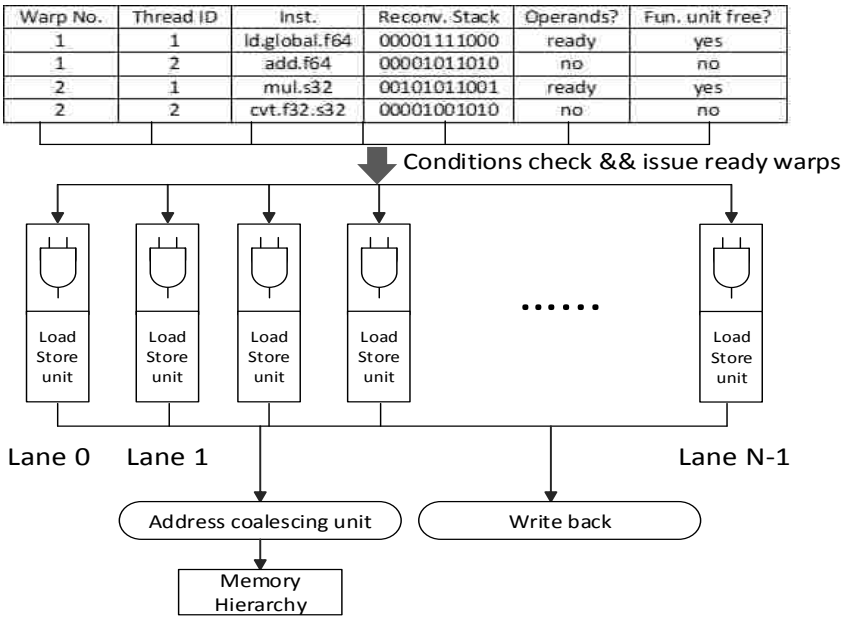


Figure 4-4. The architecture of the warp scheduler

To justify the potential effectiveness of this strategy, we run a wide spectrum of GPU applications, aiming to observe typical behaviors on the warp scheduler. Figure 4-5 plots a snapshot of the warp scheduler's behavior when *WP* is running on a GPU in order to exemplify the activity on the scheduler. The horizontal axis corresponds to the elapsed time and the vertical axis represents the accumulative number of ready warps at each time interval. The number is collected every 50 cycles. With this setting, the maximum number of ready warps cannot exceed 100 on each sampling point considering that two warp instructions can be issued at each cycle. As can be seen from the figure, there are a large amount of execution periods with number of ready warps far less than the theoretical peak, implying a significant reduction in accesses to the scheduler entries in potential. We generally observe that, at any given instant, less than 35% of

all the warps have the two prerequisites satisfied for all the tested benchmarks. This observation confirms that there is large headroom for us to optimize the reliability on the warp scheduler.

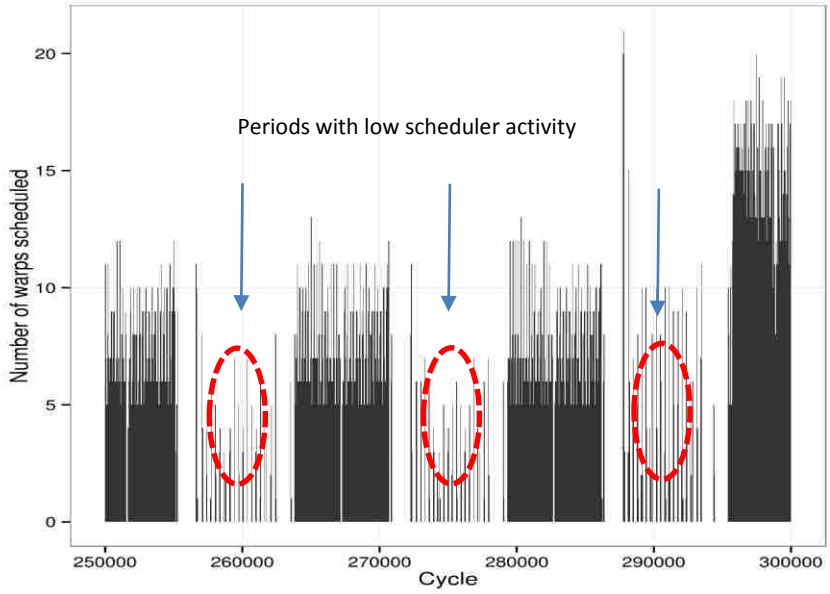


Figure 4-5. A snapshot of the scheduler activity while running *WP*

Our proposed technique to enhance the durability of the warp scheduler stems from the aforementioned fact at the first place. In order to identify the ready warps, the baseline scheduler is decoupled into two components as visualized in Figure 4-6. By doing so, the prerequisites checking is extracted from the original parallel accesses and is performed prior to obtaining the detailed information of warp instructions. This checking operation outputs the ID of all available candidates resided on the SM, triggering the consequent accesses to the hardware structure which stores all necessary information to dispatch ready warps based on the specific scheduling policy. If a large amount of resident warps are eliminated from the candidate list due to the violation of scheduling constraints, substantial accesses to the scheduler hardware (i.e., the structure at the right side in Figure 4-6) can be avoided.

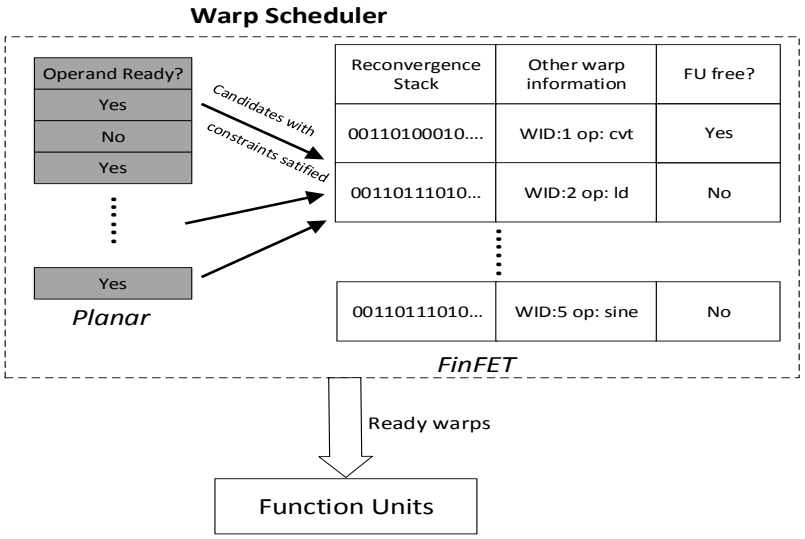


Figure 4-6. The architecture of the hybrid-device 2-stage scheduler

A non-trivial issue requiring careful consideration in this particular scheduler design is what information should be checked in the first stage. Theoretically, evaluating more scheduling pre-requisites would filter a larger number of accesses since only the common set of candidates that satisfy each individual constraint are allowed to continue to the second stage. However, for certain conditions, checking them in the first stage would lead to undesirable execution behavior because their evaluation results might be changed in the following cycle. The checking on function units' (FU) availability falls into this category. This is because the FU status is updated every cycle and a function unit that appears to be free in the current cycle is not necessarily available in the following cycle, if it is assigned to another warp instruction. Therefore in this work, we only check the data dependency in the first stage. As we will demonstrate in section 4.5, this still results in sufficiently high filter rate for most benchmarks and largely alleviates the NBTI degradation.

On the other hand, considering that the failure of any structure located on the critical path will prevent the entire chip from working correctly, the component where the condition evalua-

tions are conducted tends to become the bottleneck from the perspective of reliability, since all of its entries still need to be scanned every cycle. To overcome this problem, we propose to manufacture this component with the more NBTI-tolerable planar devices. This hybrid-device design effectively leverages the benefits of both devices, aiming to enhance the processor durability. Note that the planar-transistor-made component recording the data dependency and function unit availability is unlikely to suffer from early failure because it only requires a bit for each entry and thus consume negligible power. Also recall that this design is technically feasible due to the good compatibility between FinFET and planar processes as demonstrated in patents [20][36].

Another naturally arising concern with this design is the performance degradation resulting from the sequential scheduler access. Nevertheless, as we will demonstrate in section 4.5, the performance overhead for most applications is fairly small because only actual accesses to the FinFET part of the scheduler introduce an extra cycle delay. In scenarios where none of the resident warps pass the constraint checking, the execution latency is not impacted.

4.3.2 Hybrid-device Sequential-access L2 Cache

It is widely acknowledged by the high performance computation (HPC) community that memory bandwidth is the main bottleneck in a large number of GPU applications. Due to this reason, the shared L2 cache is becoming an increasingly important component on a modern GPU to reduce the contention on the global memory bandwidth [10], implying that improving the reliability of the L2 cache is of great significance to ensure endurable operation of the GPU.

Typically, the L2 cache installed on a contemporary GPU is designed as a set-associative cache with a reasonable size, serving memory requests sent from the stream multiprocessors. To shorten the execution delay, all ways in the tag array and data array of the selected cache set are

searched in parallel and if a stored tag equals to the tag in request, the matching cache block from the data array is returned. However, this access procedure is intrinsically unfriendly to reliable operation since it may introduce substantial unnecessary cache accesses in case the requested data block is not present. For example, the application *Blackscholes* demonstrates a close-to-100% miss rate on the L2 cache, meaning that approximately all the memory requests that are missed in the L1 cache need to be transferred to the global memory eventually. In other words, accesses to the L2 cache are completely unnecessary.

Based on this observation, it is straightforward to realize that filtering out the accesses resulting in cache misses is a simple yet effective approach to slow down the NBTI aging on the L2 cache. Since the data array is orders of magnitude larger than the tag array in both area and power consumption, we first concentrate on the optimization of the data array, which is achieved by applying a technique similar to that developed for the warp scheduler. Specifically, we serialize the parallel tag/data access into a sequential procedure [31] in which the tag array in the selected cache set is probed first and only in case a matching tag is found, is the corresponding block in the data array accessed. This particular design, as visualized by Figure 4-7, reduces the accesses to the data array in two-manners, (1) memory requests that result in cache misses (i.e., no matching tag is found) do not generate consequent accesses to the data array, and (2) only the cache block corresponding to the matching tag, instead of all ways in the set, is read to respond the memory request. With this technique, we expect that the accesses to the data array should be considerably reduced, thus the NBTI aging is largely suppressed due to the decreasing activity and temperature.

On the other hand, to prevent the tag array from becoming the reliability bottleneck, we exploit the device heterogeneity and propose to build the tag array with planar transistors. As we

will show in later sections, this can effectively leverage the planar device's advantage in NBTI-tolerance and guarantee reliable operations on the L2 tag array throughout the expected lifespan. Also note that in the remainder of this chapter, we may interchangeably use the terms planar-tag L2, hybrid-device L2, and sequential-access L2 to refer to this design.

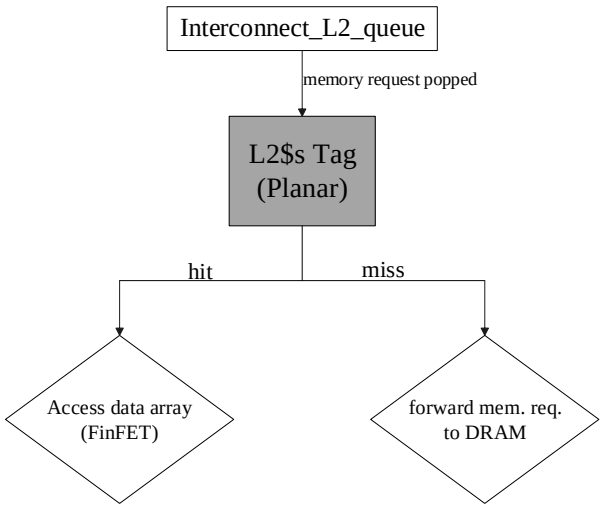


Figure 4-7. Workflow of the hybrid-device sequential-access L2 cache

4.4 Experimental Setup

We validate the proposed techniques using a modified GPGPU-Sim 3.1 [23]. GPUWattch [68] and HotSpot 5.0 [16] are integrated in the simulator for power and temperature calculation, respectively. The chip floorplan required by HotSpot is calibrated against the one used in a recent paper focusing on GPU thermal management [79]. The target architecture is configured based on a Fermi GTX 480 [12] that is widely used in many high-performance computers. Table 4-1 lists the detailed architectural parameters for our simulation.

To evaluate the effectiveness of our techniques in practice, we choose a set of programs from several benchmark suites [10][23][29], representing typical HPC applications derived from different domains. A full list of applications used in this work is given in Table 4-2. For each

program, we run them till completion and use the execution statistics to mimic distinct workload patterns. Specifically, to model the NBTI degradation after a 7-year lifespan, we extrapolate the collected activity to represent the load in 7 years under steady temperature. We report the final increase in the critical path delay as a measurement of the NBTI aging on the hardware.

Table 4-1. Architectural parameters for the GPU in study

Parameter	Values
#SM	15
#SP	32/SM
LDST units	16/SM
Shared memory	32KB/SM
L1 data cache	16KB/SM
Scheduler	Greedy than oldest (GTO)
Core frequency	1400MHz
Interconnection	1 crossbar/direction
L2 cache	768KB: 128 cache line size, 16-way associativity. Access latency 5 cycles
L2 frequency	700MHz
Memory	FR-FCFS scheduling, 64 max. requests/MC
SIMD lane width	16
Threads/warp	32
Technology	22nm

Equations 4.2 and 4.3 described in section 4.2.1 are used to compute the variation in the threshold voltage, which in turn translates to the delay increase via equation 4.1. We set the parameters referred by the equations according to recent studies on device features [17][28][97]. Table 4-3 lists the specific parameter values used in this chapter.

4.5 Result Analysis

In this section, we demonstrate the experimental results corresponding to each technique and analyze them in detail. We first demonstrate the improvement in mitigating NBTI degradation when our techniques are adopted and then present the performance overhead afterwards.

Table 4-2. Benchmarks used in this work

#	Application	Domains
1	B+tree	Search
2	Backprop	Pattern Recognition
3	Barneshut	N-body Simulation
4	BFS	Graph Algorithms
5	Blackscholes	Financial Engineering
6	Gaussian	Linear Algebra
7	Heartwall	Medical Imaging
8	Hotspot	Physics simulation
9	LavaMD	Molecular Dynamics
10	LPS	3D Laplace Solver
11	Myocyte	Biological Simulation
12	NN	Neural Network
13	NQU	N-Queen Solver
14	NW	Bioinformatics
15	WP	Weather Prediction

Table 4-3. Parameter values for computing NBTI

Parameters	FinFET value	Planar value	Description
T_{ox}	1.2nm	1nm	Effective oxide thickness
V_t	0.179v	0.3v	Threshold voltage
E_o	0.335v/nm	0.12v/nm	Electrical field
Fixed parameters			
q	1.602×10^{-19}		Electron charge
V_{dd}	0.9v		Operating voltage
ε_{ox}	1.26×10^{-19}F/m		Permittivity of gate oxide
ξ_1	0.9		Other constants
ξ_2	0.5		
k	8.6174×10^{-5} ev/K		
δ	0.5		
T0	10^{-8} s/nm^2		

4.5.1 Warp Scheduler

Figure 4-8 demonstrates the NBTI degradation in terms of the increase in scheduler delay on both the baseline GPU and the one with hybrid-device 2-stage warp scheduler. Note that in the figure, the bars marked by "2-stage" refer to the proposed design. A higher delay increase indicates more severe NBTI degradation. As can be seen from the diagram, the aging due to NBTI

on the scheduler hardware is largely suppressed for all benchmarks under investigation when the proposed technique is applied. On average, the hybrid-device 2-stage scheduler presents merely 2.4% longer delay after the designed service life, reduced from 7.5% on the baseline GPU.

While the general improvement on the durability is significant, however, it is notable that the benefits corresponding to different workloads are obviously distinct. For example, the load represented by *NN* causes the scheduler delay to be prolonged by around 8.4% after 7 years services on the baseline GPU. With the adoption of the proposed technique, this degradation can be reduced to 1.96%. On the other hand, an execution pattern similar to *Backprop* prevents the scheduler obtaining the same amount of benefit from the technique. Specifically, the scheduler still suffers from 2.9% longer delay after employing the hybrid-device design, while the baseline platform shows 8.6% longer delay that is similar to the degradation corresponding to *NN*.

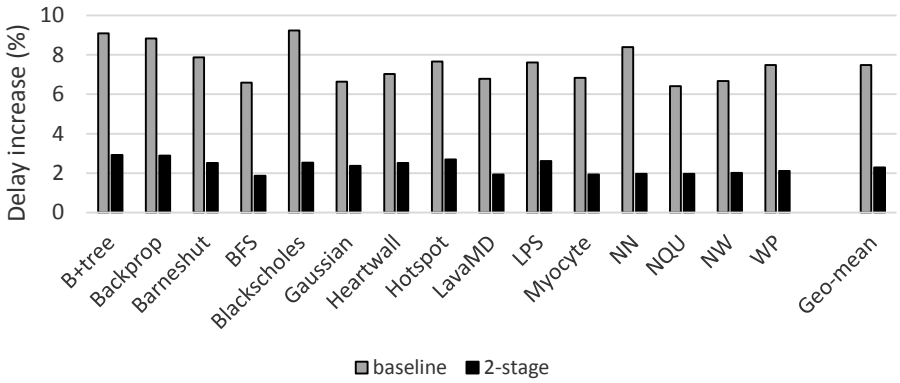


Figure 4-8. The NBTI degradation on the warp scheduler

Considering the exponential relationship between temperature and NBTI degradation, we collect the localized temperature on the scheduler hardware and demonstrate it in Figure 4-9 for further analysis. Not surprisingly, although the proposed technique can significantly cool down the scheduler in most cases, we note that the temperature reductions are apparently different among the evaluated programs, which is similar to the observation made from Figure 4-8. When

executing *NN*, the temperature on the scheduler is reduced by up to 15 ℃, whereas the temperature reduction for *Backprop* is less than 12 ℃. To gain more insights into the reason behind this phenomenon, let us recall the rationale of the 2-stage scheduler that is described in section 4.3.2. The essential reason for the reduced scheduler accesses is that a large amount of prerequisite evaluations turn out to be false, thus the unnecessary operations on the "unready warps" are avoided. In other words, how much benefit can be obtained from the proposed technique largely depends on the amount of accesses that can be filtered. Table 4-4 lists the percentage of accesses saved by the constraint checking stage. As can be seen, the data dependency checking stage can generally filter out more than 92% of accesses to the scheduler, thus considerably enhancing the durability of the hardware. In particular, we note that 76.9% of scheduler accesses when executing *Backprop* are dispensable, while for *NN* this ratio rises up to 97.4%, implying higher possibilities to lower the power and temperature on the scheduler.

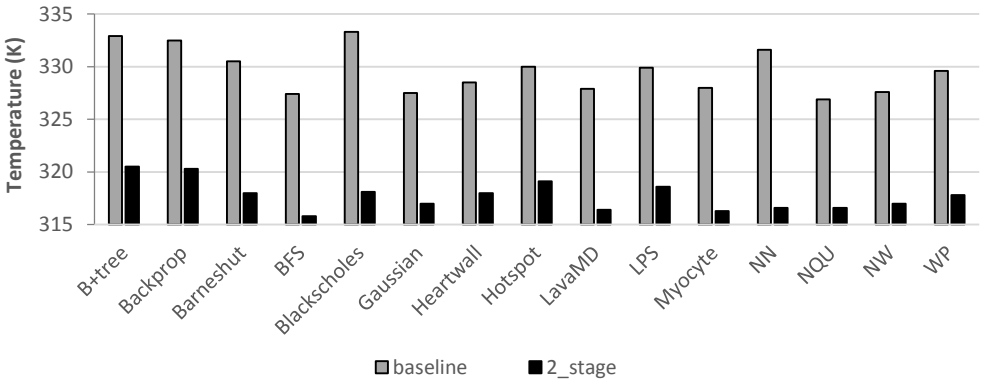


Figure 4-9. The steady temperature on the warp scheduler

We also plot the power consumption of the scheduler in Figure 4-10 in order to visualize the changes on the scheduler activity. Clearly, the hybrid-device 2-stage scheduler significantly reduces the scheduler power for all evaluated benchmarks, which in turn lowers the localized temperature and improves the hardware durability.

Table 4-4. Filter rate on the first stage of warp scheduler

Application	Filter Rate
B+tree	75.82%
Backprop	76.93%
Barneshut	91.55%
BFS	98.17%
Blackscholes	88.74%
Gaussian	98.82%
Heartwall	88.46%
Hotspot	86.5%
LavaMD	99.21%
LPS	90.59%
Myocyte	99.85%
NN	97.41%
NQU	98.2%
NW	97.70%
WP	99.49%
Geo-mean	92.15%

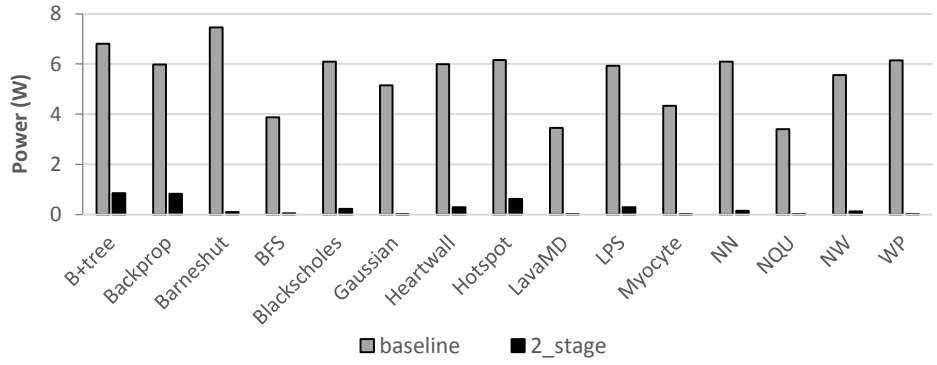


Figure 4-10. The power consumed by the warp scheduler

The extra cycle introduced by the 2-stage scheduler is likely to result in undesirable performance overhead for the program execution. Figure 4-11 shows the performance in terms of normalized IPC (normalized to the baseline GPU) of all benchmarks running on a GPU with the 2-stage scheduler. It is straightforward to note that the performance degradation is distinct among the program collection. In this subsection, we briefly analyze the possible impact on the performance due to the extra cycle and explain the different performance degradation.

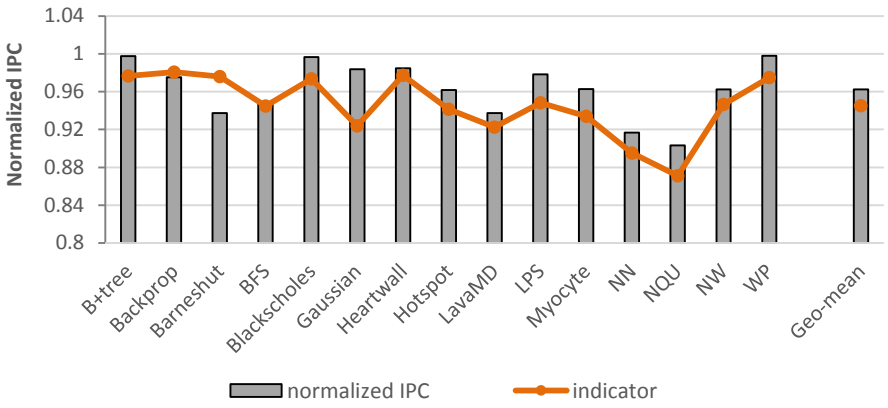


Figure 4-11. Normalized IPC on the GPU with 2-stage scheduler

The GPU's massive parallelism may be able to hide part of the extra latency during the execution depending on the features of applications. We use the terms "longest warp" and "longest-warp chain" to help explain the latency manifested in the results. We define "longest warp" as the warp with the longest running time during a kernel launch and "longest-warp-chain" as the set of longest warps in each of the sequence of kernel launches in the lifetime of an application. In a typical GPU application, the running time of a longest-warp chain is the sum of execution latencies of all warps in the chain because a) when a kernel is launched, all its warps are started simultaneously and b) a kernel is not launched until all warps of the previous kernel launch complete. In other words, latency on the longest warp could not be hidden as easily as that on other warps. Longest warps also do not overlap temporally. For each longest warp we can compute its average latency as:

$$AvgLatency = \frac{\sum_{n=1}^{N} C_n}{\sum_{n=1}^{N} I_n} \quad \text{4.4}$$

where N is the number of kernel launches and C_n and I_n are respectively the number of cycles and warp instructions of the longest warp in each kernel launch.

The I_n instructions in a kernel launch are the instructions issued to and executed by a warp. The extra cycle introduced to the scheduler will be added before each of the instructions is executed. Since the instructions are executed in-order, this is equivalent to adding $\sum_{n=1}^{N} I_n$ extra cycles to the entire longest-warp chain. The average latency of the warp after adding the extra cycles should become:

$$AvgLatency_{delayed} = \frac{\sum_{n=1}^{N} C_n + \sum_{n=1}^{N} I_n}{\sum_{n=1}^{N} I_n} \quad \text{4.5}$$

The overhead indicators can be deducted from the two latencies shown above:

$$OverheadInd = \frac{\Delta latency}{AvgLatency} = \frac{AvgLatency_{delayed} - AvgLatency}{AvgLatency} = \frac{\frac{\sum_{n=1}^{N} C_n + \sum_{n=1}^{N} I_n}{\sum_{n=1}^{N} I_n} - \frac{\sum_{n=1}^{N} C_n}{\sum_{n=1}^{N} I_n}}{\frac{\sum_{n=1}^{N} C_n}{\sum_{n=1}^{N} I_n}} =$$

$$\frac{\sum_{i=1}^{N} C_n + \sum_{i=1}^{N} I_n - \sum_{i=1}^{N} C_n}{\sum_{i=1}^{N} C_n} = \frac{\sum_{i=1}^{N} I_n}{\sum_{i=1}^{N} C_n} = \frac{1}{AvgLatency} \quad \text{4.6}$$

The normalized IPC and the one derived from the overhead indicator are both plotted in Figure 4-11. As the figure shows, they are closely correlated. The average latencies and the overheads are determined by the behaviors of the longest warps which are in turn closely related to the characteristics of individual applications. For example, *B+tree* involves a kernel launch with 48 warps on each SM and initiates many global memory transactions (159.26 per cycle). Its longest warp has an average delay of more than 100 cycles. NQU, on the other hand, has a much smaller average delay (smaller than 10), because it generates much fewer global memory transactions (only 0.018 per cycle) and each SM executes only 8 warps. With such few memory transactions and fewer warps, each of the warps, including the longest warp, does not have to wait for long-delay memory operations while sharing more computational resources. These different memory request intensities result in average latencies of the longest warp chains as 41.7

and 6.76 cycles for *B+tree* and *NQU*, respectively. Consequently, we observe apparently different performance losses for these two benchmarks.

As shown in Figure 4-11, on average, the overhead is less than 4% across the benchmark collection. Therefore, based on the evaluation results, it is safe for us to conclude that the hybrid 2-stage scheduler is effective in significantly enhancing the device's durability with mild performance overhead. Obviously, this corroborates the benefit of exploiting device heterogeneity in future processors made of emerging transistor devices.

4.5.2 L2 Cache

We now shift our concentration to the L2 cache. For this structure, we first focus on its data array. Figure 4-12 shows the NBTI degradation on the L2 cache data array on both the baseline GPU and the GPU with a planar-tag sequential-access L2. Note that the latter one is labeled as "with_Ptag" in the figure, where the capital letter P stands for planar device. As shown in the figure, the general trend is similar to what is observed in previous section that the proposed technique is capable of largely slowing down the aging due to NBTI on the target component throughout the service life. On average, the hybrid-device design reduces the delay increase from 6.1% in the baseline situation to 2.1%.

We also note that the improvement on the durability is different among the programs in study. For example, the applications *Barneshut* and *LPS* cause approximately the same level of NBTI aging on the baseline platform. However, with the hybrid-device L2 cache, running *LPS* apparently leads to less significant NBTI degradation (2.1%) compared to the execution of *Barneshut* (2.8%). This is resulted from the distinct temperature variations on the L2 while running these programs. Figure 4-13 shows the steady L2 temperature for both the baseline and our

proposed design. From the figure, we note that on the GPU with the hybrid-device design, running LPS makes the L2 cache much cooler compared to the execution of *Barneshut*. The reason is as follows. Similar to accessing the 2-stage warp scheduler, memory requests sent to the L2 cache are served in a sequential tag-data access pattern, while the tag probing can eliminate the unnecessary accesses to the data array (i.e., cache misses). In other words, the different amount of cache accesses that are avoided are the essential reason for the distinct temperature and reliability changes.

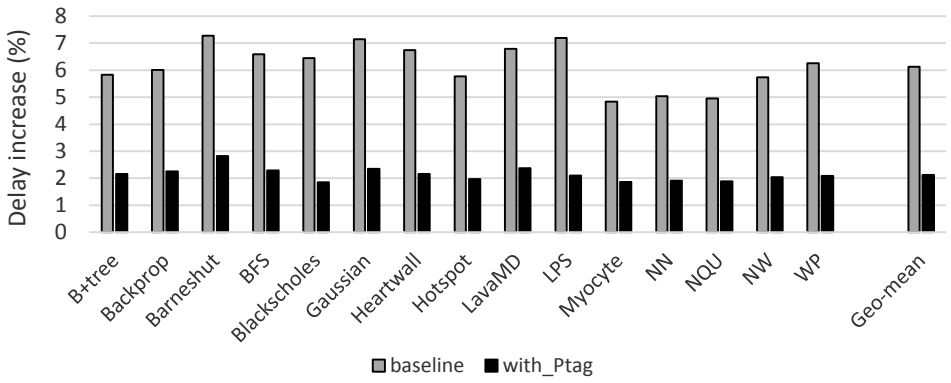


Figure 4-12. NBTI degradation on the L2 data array

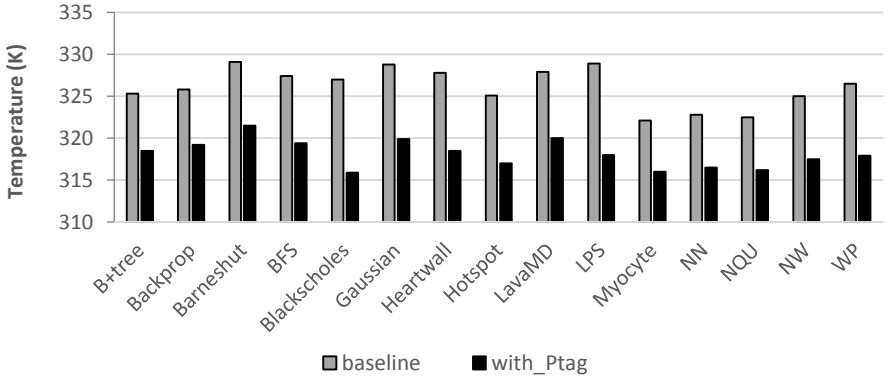


Figure 4-13. Steady temperature on the L2 data array

Figure 4-14 and Figure 4-15 respectively plot the L2 cache miss rates and comparison of L2 power for different applications. As can be seen, *LPS* demonstrates an L2 miss rate of 36%, thus

resulting in impressive reduction in L2 power/temperature and great reliability enhancement as a consequence. For *Barneshut*, most of the accesses to the data array cannot be avoided because of the low L2 miss rate (4.7%). This eventually leads to the relatively smaller improvement on the NBTI degradation. Other benchmarks with high L2 miss rates including *Blackscholes* also present relatively larger improvement on device durability compared to those with low L2 miss rates such as *NN*. On the other hand, it is important to keep in mind that even for a cache hit, only the matching block is accessed afterwards. For caches with high associativity, which is the typical design in many modern processors, this provides another fold of reduction in the localized power and temperature. Due to this reason, the power consumption of L2 for all benchmarks is considerably reduced while running with sequential-access cache as shown in Figure 4-15.

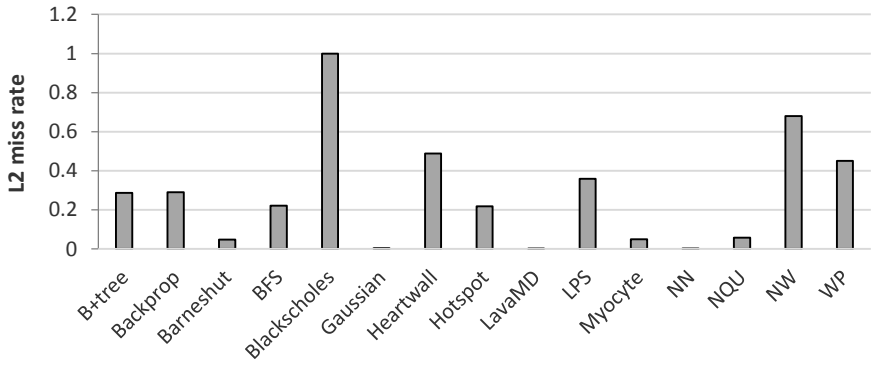


Figure 4-14. L2 miss rate of all evaluated benchmarks

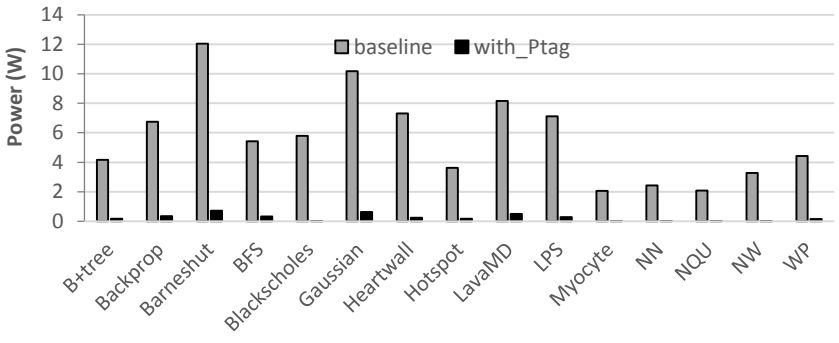


Figure 4-15. Power consumption of the L2 data array of all evaluated benchmarks

The reliability of the tag array is becoming a major concern in the proposed cache design since the accesses to this structure have not been reduced. Fortunately, due to higher NBTI-immunity manifested by planar transistors and the small power consumed by the tag array, the L2 tag is not likely to suffer from significant NBTI degradation. Figure 4-16 compares the NBTI degradation in the tag array with both designs, which is essentially determined by the different NBTI tolerance of FinFET and planar transistors. As can be seen, the tag array made of planar device leads to much less degradation compared to the baseline platform, implying more endurable operation in the service life.

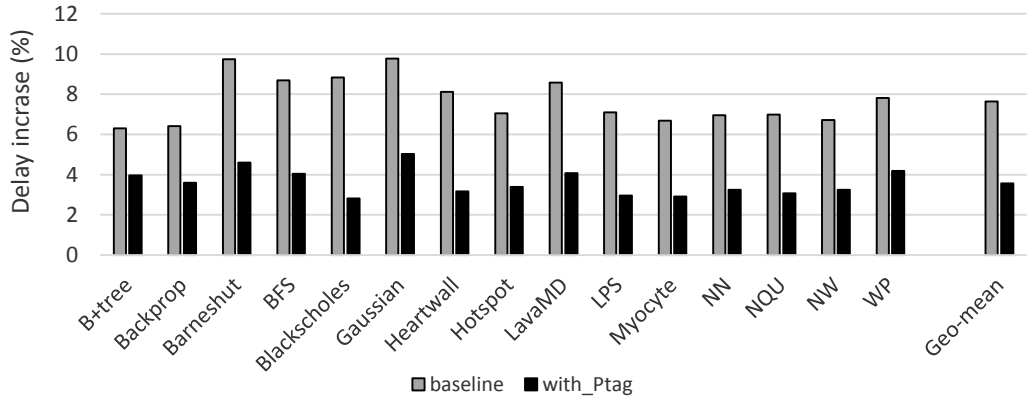


Figure 4-16. NBTI degradation on the L2 tag array

Another concern that deserves evaluation is the possible performance loss resulting from the extra delay spent on the cache tag probing. We demonstrate the normalized IPC of all programs with the sequential-access L2 cache in Figure 4-17 and find that the performance degradation for all benchmarks in investigation is within 1.5%. This does not go beyond our expectation due to the following reasons. First, only a cache hit introduces an extra cycle delay since misses will be promptly forwarded to the lower memory hierarchy after the tag probing, thus not wasting any cycles. Second, even an L2 cache hit takes multiple cycles to complete. This includes the 5 cycles to access the data array and the time spent on the interconnection network. Therefore, the

extra one cycle does not weigh heavily and will not evidently impair the overall performance. Figure 4-18 plots the average L2 hits per cycle (i.e., actual accesses to the data array) for the program collection in order to briefly explain the different impacts on the performance caused by the extra cycle. As can be observed, applications such as *Blackscholes*, *Myocyte*, *NQU* and *NW* have extremely low L2 hits intensity, so their performance is not notably degraded (close to zero loss) with the sequential-access L2 cache. On the contrary, *Barneshut* and *Gaussian* result in more frequent L2 hits, thus their execution speed is lowered by a relatively higher percentage (1.5%). Nonetheless, based on the evaluations made on the L2 cache, it is still reasonable for us to conclude that the proposed hybrid-device sequential-access design can significantly slow down the NBTI aging on the L2 cache with slight performance overhead.

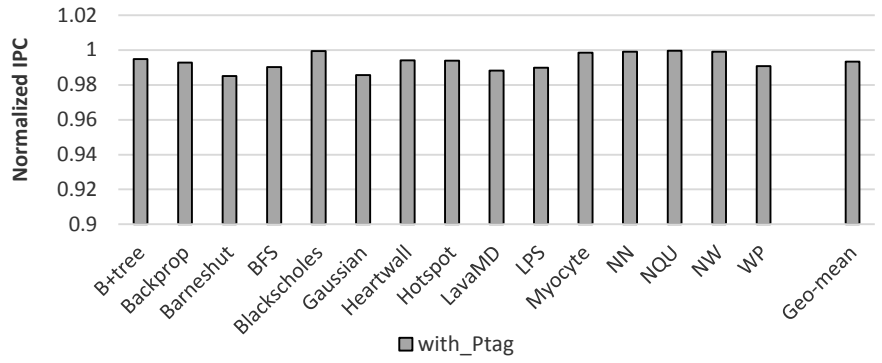


Figure 4-17. Normalized IPC with the sequential-access L2 cache

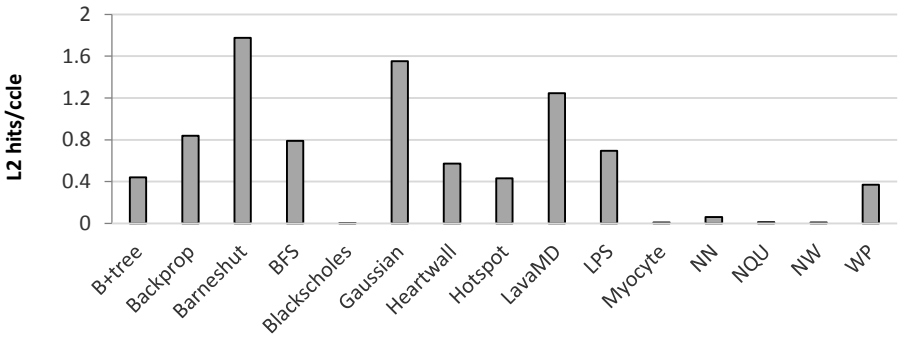


Figure 4-18. L2 hits/cycle of all evaluated benchmarks

4.6 Related Work

On the first aspect, NBTI has been recognized as a major reliability concern as the semiconductor industry shifts into the deep submicron era. To mitigate the NBTI degradation and enhance the device's durability, researchers have conducted substantial works in the past years. Abella et al. [18] develop a set of techniques to relieve the NBTI aging for typical structures in a modern CPU. For combinational logic, they insert desired vectors as inputs to the structures for recovery. To alleviate the aging for memory-like components, they propose a strategy to avoid the bias on different bits. Ramakrishnan et al. [85] introduce a similar approach to reduce the NBTI wearout in FPGAs by loading the reversing bit patterns in idle periods. Gunadi et al. [47] introduce a scheme called Colt to balance the utilization of devices in a processor for reliability improvement. Specifically focusing on the storage components, Shin et al. [95] propose to proactively set the PMOS transistors to recovery mode and move data around free cache arrays during operation.

Converse to these works which attempt to manipulate the time under stress and recovery, Tiwari et al. [106] propose a framework named facelift to combat NBTI degradation by adjusting higher level parameters including operating voltage, threshold voltage and the application scheduling policy. Fu et al. [42] concentrate on the NBTI mitigation in presence of process variation. They effectively utilize the interplay between NBTI aging and process variation to prevent early failure of specific structures. To enhance the reliability of storage cells, Abella [19] proposes to use NAND gates instead of inverters to reduce the average degradation on each PMOS.

There are few works aiming to alleviate the NBTI aging on GPUs in the literature. Rahimi et al. [84] focus on the GPUs designed in VLIW fashion and present a technique to slow down the NBTI aging for this particular architecture. By exploring the unbalanced usage among function

units within a VLIW slot, their proposed strategy can uniformly assign the stress among all computation units and achieve an even aging rate.

On the second aspect, as FinFET is widely considered as an attractive replacement of planar transistors for the next few technology nodes, studies focusing on the reliability of this new structure are becoming fairly important. Lee et al. [67] investigate the NBTI characteristics of SOI and body-tied FinFETs and observe that a narrow fin width leads to more severe degradation than a wider fin width. Crupi et al. [34] compare the reliability of triple-gate and planar FETs. The author show that the behavior of time-dependent dielectric breakdown (TDDB) is not changed on the triple-gate architecture under different gate voltages and temperatures. This is also corroborated in the work conducted by Groeseneken et al. [46], which further demonstrates that FinFET devices tend to suffer from more severe NBTI degradation. In [109], Wang et al. analyze the soft-error resilience of FinFET devices and conclude that a FinFET circuit is more reliable than a bulk CMOS circuit in terms of soft-error immunity.

Finally, on the third aspect, exploiting device-level heterogeneity has been widely used for performance and energy efficiency optimization in computer architecture study. Saripalli et al. [92][93] discuss the feasibility of technology-heterogeneous cores and demonstrate the design of mixed-device memory. Wu et al. [113] present the advantage of hybrid-device cache. Kultursay [62] and Swaminathan [102] respectively introduce a few runtime schemes to improve performance and energy efficiency on CMOS-TFET hybrid CMPs. For the optimization on GPUs, Goswami et al. [44] propose to integrate resistive memory into the compute core for reducing the power consumption on GPU register file.

Our work deviates from the aforementioned studies in that we aim to alleviate the NBTI degradation of GPUs made of FinFET from the architectural level. To the best of our knowledge, this work is the first attempt to address this increasingly important problem.

4.7 Conclusion

FinFET technology is recognized as a promising substitute of conventional planar devices for building processors in the next decade due to its better scalability. However, recent experimental studies demonstrate that FinFET tends to suffer from more severe NBTI degradation compared to the planar counterpart. In this work, we focus on the NBTI reliability issue of a modern GPU made of FinFET and propose to address this problem by exploiting the device heterogeneity. We introduce a set of techniques that merely involve minor modifications to the existing GPU architectures. The proposed techniques leverage planar devices' higher immunity to NBTI and are effective in slowing down the aging rate of the device. Our evaluation results demonstrate that the minor changes to the warp scheduler and the L2 cache can considerably alleviate the degradation due to NBTI with slight performance overhead.

CHAPTER 5. SUMMARY AND FUTURE WORK

5.1 Summary

The persistent pursuit of building faster processors with smaller and power-saving transistors has driven the development of integrated circuit in the past decades. However, the increasingly important power-wall issue, along with the physical limitation of conventional devices, has posed a significant challenge to the semiconductor industry which aims at continuing the fast processor evolution in the ensuing decade. To overcome this daunting conundrum, researchers have proposed several solutions that can alleviate the negative impacts due to those difficulties. Among all the solutions, exploiting heterogeneity in different stages of processor manufacturing and operation is widely acknowledged as a promising one. While the diversities in processor architecture and manufacturing materials provide opportunities for exploiting the heterogeneity, they also introduce a set of unprecedented challenges that requires effective solution. This research presents a series of studies addressing these problems.

First, we explore a vast design space in order to select the most promising configurations for future chip multi-processors when both architectural- and device-heterogeneity are taken into account. By comprehensively investigating the impact of varying configurations on important design goals including performance, energy-efficiency and cost-efficiency, we corroborate the advantage of architectural asymmetry and mixed-device processors over their homogeneous counterpart. Moreover, we propose the concept of two-fold heterogeneity with which processor cores of different architectures are made of distinct materials. The evaluation results demonstrate that such a design paradigm appears to be the most attractive one for future CMPs as it effectively leverage the benefits of both heterogeneities.

Second, we focus on single-ISA heterogeneous CMP platforms and aim at addressing the scheduling issue on such hardware. Implementing a heterogeneity-aware task scheduler is one of the key problems in studies related to heterogeneity because the effectiveness of the scheduler essentially determines how much benefit can be obtained from the underlying asymmetric hardware. While there are already some works concentrating on this issue in the literature, we address it from a different perspective. In particular, we observe that scheduling decisions which maximize the throughput do not necessarily lead to the most energy-saving execution. Based on this fact, we propose to employ an advanced statistical tool to extract a set of simple "IF-ELSE" conditions to guide the dynamic task scheduling. The proposed technique demonstrates good scalability and is fairly easy to implement. Our evaluation results prove that the rule-set guided scheduler can effectively decrease the total energy consumption while delivering similar performance to the existing performance-aimed schedulers.

The third study presented in this research pays attention to the device heterogeneity. While FinFET has been considered as an attractive substitute for conventional planar transistors for building processors in the sub-32nm era, recent experimental studies show that FinFET tends to suffer from more serious NBTI degradation compared to its planar predecessor. Taking this into consideration, we propose to mitigate the NBTI degradation on many-core (GPU) processors made with FinFET through exploiting the device heterogeneity. Specifically, we demonstrate that the activity of some important components on representative GPUs can be largely degraded by small filter-like structure. In this situation, using small filters made with NBTI-tolerable planar transistors to reduce the accesses to FinFET main components is a straightforward solution to slowing down the NBTI aging on the FinFET structures. We present hybrid-device designs of two important components (i.e., warp schedulers and L2 cache) on top of this principle and

demonstrate that this technique is effective in alleviating the NBTI degradation with slight performance overhead for FinFET-made GPUs.

5.2 Future Work

While the research presented in this dissertation focuses on both CPU and GPU, the fused CPU-GPU platform (e.g., the APU released by AMD), which also serves as another important heterogeneous architecture, is not discussed here. In the near future, I will make a concentrated shift to this platform and conduct further investigations. With the prevalence of cross-platform programming languages such as OpenCL, it becomes possible for programmers to implement general-purpose applications on the GPU where the execution can benefit from massive thread-level parallelism (TLP). Therefore for these programs, appropriate GPU implementations demonstrate dramatic performance improvement compared to the corresponding CPU version. As a consequence, GPUs are now widely equipped in high-performance computers to boost the computation capability. On a CPU-GPU heterogeneous platform, nevertheless, this implies that the integrated graphics processing unit can be used concurrently with the CPU for program execution and improve the performance.

On the other aspect, similar to existing chip multi-processors, the CPU-GPU architecture requires appropriate management for the shared resource in order to deliver the optimal performance or energy-efficiency. Resources that are shared between the two components depend on specific processor architectures. For example, on an AMD APU, the CPU and GPU have their dedicated cache hierarchies and only the memory controller is shared; in contrast, an Intel Ivy Bridge processor includes a shared last-level cache (LLC) in addition to the memory controller between two components. Nevertheless, care should be taken when both the CPU and GPU are running programs since contention on the shared cache or memory bandwidth may significantly

impair the performance of a single program. Therefore, shared resource management and allocation is of great importance for the performance optimization on CPU-GPU architectures.

Based on the above analysis, we will conduct the following potential research work on the CPU-GPU platform.

The first study is on the workload partitioning. As the GPU is integrated on the same die where the CPU is installed, the overhead due to communications between the two components has been minimized and thus the CPU and GPU can cooperate with each other more tightly. Taking this into consideration, it is reasonable to divide a task and assign appropriate shares of work amounts to the two computing units for execution. Obviously, such a strategy yields more effective usage of the system computation resources and accordingly delivers better performance. To achieve this goal, a mechanism which determines the optimal workload distribution between the CPU and GPU is in high demand. Overestimating the computing capability of one side may lead to an unbalanced task assignment and the relatively slow device becomes the bottleneck as a consequence. The statistical tools introduced in section 3.2 can be employed to address this problem. Namely, we train a model to bridge the gap between the execution behaviors from both sides and the total run time. After doing this, when a new program (or a new iteration of an existing program) is ready for scheduling, the model predicts the most appropriate amount of work given to the CPU and GPU, thus delivering the optimal performance.

The second investigation will focus on the shared resource. The management of shared last-level cache and memory bandwidth has been extensively discussed in the context of conventional chip multiprocessors. On the emerging CPU-GPU platform, this becomes even more important because of the distinctive execution behaviors of programs running on the GPU. As described

earlier, applications running on the graphics processing unit generally spawn a large number of concurrent threads, implying substantial memory requests to the shared LLC and DRAM. Nevertheless, this does not necessarily mean that more cache space and memory bandwidth should be reserved for the GPU. The reason is that general-purpose programs running on a GPU are able to hide the memory latency via thread-level parallelism; in this situation, blindly increasing the share of cache space or DRAM bandwidth to the GPU may significantly degrade the performance of programs running on the CPU, while delivering slight performance benefit to the GPU application. As a consequence, we need to investigate the impact of resource contention on the performance of the CPU and GPU, respectively, after which we are capable of developing a mechanism that appropriately manages the shared resource for optimal performance and desirable quality of service.

REFERENCES

[1] AMD Corporation. AMD Accelerated Processing Units. http://www.amd.com/us/products/technologies/fusion/Pages/fusion.aspx.

[2] ARM Corporation. Big.Little Processing. http://www.arm.com/products/processors/technologies/bigLITTLEprocessing.php.

[3] Intel Corporation. 3rd Generation of Intel Core i7 Processor. http://ark.intel.com/products/family/65505.

[4] Intel Corporation. 4th Generation of Intel Core i7 Processor. http://ark.intel.com/products/family/75023.

[5] Intel Corporation. High-K and Metal Gate Transistor Research. http://www.intel.com/pressroom/kits/advancedtech/doodle/ref_HiK-MG/high-k.htm.

[6] Intel 64 and IA-32 architectures software developers manual volume 3: system programming guide. Oct. 2011.

[7] Intel Corporation. Intel's revolutionary 22nm transistor technology. May 2011.

[8] International Technology Roadmap for Semiconductors. http://www.itrs.net/.

[9] Modified SPLASH-2 benchmarks. http://www.capsl.udel.edu/splash/.

[10] Nvidia Corporation. CUDA C Programming Guide.

[11] Nvidia Corporation. CUDA Computing SDK 4.2.

[12] Nvidia Corporation. GTX 480 Specifications. http://www.geforce.com/hardware/desktop-gpus/geforce-gtx-480/specifications.

[13] Nvidia Corporation. Tegra 3 super chip processors. http://www.nvidia.com/object/tegra-3-processor.html.

[14] Standard Performance Evaluation Corporation. SPEC CPU 2006.

[15] http://www.eetimes.com/document.asp?doc_id=1264668.

[16] Hotspot 5.0 Temperature Modeling Tool. http://lava.cs.virginia.edu/HotSpot.

[17] Predictive Technology Model. http://ptm.asu.edu.

[18] J. Abella, X. Vera, and A. Gonzalez. Penelope: The NBTI-aware processor. In *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Dec. 2007.

[19] J. Abellea, X. Vera, O. Unsal, and A. Gonzalez. NBTI-resilient memory cells with NAND gates for highly-ported structures. In *Workshop on Dependable and Secure Nanocomputing*, Jun. 2007.

[20] B. A. Anderson, A. J. Joseph, and E. J. Nowak. Integrated circuit including FinFET RF switch angled relative to planar MOSFET and related design structure. U.S. Patent 8125007 B2, Feb. 2012.

[21] A. Asenov, C. Alexander, C. Riddet, and E. Towie. Predicting future technology performance. In *Proceedings of 50th Design Automation Conference (DAC)*, Jun. 2013.

[22] S. Balakrishnan, R. Rajwar, M. Upton, and K. Lai. The impact of performance asymmetry in emerging multicore architectures. In *ISCA*, Jun. 2005.

[23] A. Bakhoda, G. Yuan, W. Fung, H. Wong, and T. Aamodt. Analyzing CUDA Workloads Using a Detailed GPU Simulator. In *ISPASS*, 2009.

[24] M. Becchi and P. Crowley. Dynamic Thread Assignment on Heterogeneous Multiprocessor Architectures. In *CF*, May 2006.

[25] C. Bienia, S. Kumar and K. Li. PARSEV vs. SPLASH-2: a quantitative comparison of two multithreaded benchmark suites on chip-multiprocessors. In *IISWC*, 2008.

[26] C. Bienia, S. Kumar, J. P. Singh and K. Li. The PARSEC benchmark suite: characterization and architectural implications. In *PACT*, Oct. 2008.

[27] Breiman, L., Friedman, J.H., Olshen, R.A., and Stone, C.J. Classification and Regression Trees, Wadsworth Press, 1984.

[28] S. Chaudhuri, and N. K. Jha. 3D vs. 2D analysis of FinFET logic gates under process variations. In *ICCD*, Oct. 2011.

[29] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, and K. Skadron. Rodinia: A Benchmark Suite for Heterogeneous Computing. In *IISWC*, Oct. 2009.

[30] J. Chen and L. K. John. Efficient program scheduling for heterogeneous multi-core processors. In *DAC*, Jun. 2009.

[31] Z. Chishti, M. D. Powell, and T. N. Vijaykumar. Distance associativity for high performance energy efficient non-uniform cache architectures. In *Proceedings of International Symposium on Microarchitecture (MICRO)*, Dec. 2003.

[32] J. P. Colinge, "Multiple-gate SOI MOSFETs", *Solid-State Electronics*, vol. 48, no. 6, June 2004.

[33] K. V. Craeynest, A. Jaleel, L. Eeckhout, P. Narvaez, and J. Emer. Scheduling heterogeneous multi-cores through performance impact estimation. In *ISCA*, Jun. 2012.

[34] F. Crupi, B. Kaczer, R. Degraeve, V. Subramanian, P. Srinivasan, E. Simoen, A. Dixit, M. Jurczak, and G. Groeseneken. Reliability comparison of triple-gate versus planar SOI FETs. In *IEEE Transactions on electron devices*, vol. 53, no. 9, Sept. 2006.

[35] H. F. Dadgour and K. Banerjee. Design and analysis of hybrid NEMS-CMOS circuits for ultra low-power applications. In *DAC*, Jun. 2007.

[36] B. B. Doris, D. C. Boyd, M. Leong, T. S. Kanarsky, J. T. Kedzierski, M. Yang. Hybrid planar and FinFET CMOS devices. U.S. Patent 7250658 B2, Jun. 2007.

[37] R. G. Dreslinski, M. Wieckowski, D. Blaauw,D. Sylvester, and T.Mudge. Near-threshold computing: reclaiming Moore's law through energy efficient circuit. In *Proceedings of the IEEE, special issue on ultra-low power circuit technology*, Feb. 2010.

[38] H. Esmaeilzadeh, E. Blem, R. St. Amant, K. Sankaralingam, D. Burger. Dark silicon and the end of multicore scaling. In *ISCA*, Jun. 2011.

[39] H. Esmaeilzadeh et al. Addressing dark silicon challenges with disciplined approximate computing. In *Dark Silicon Workshop in conjunction with ISCA*, Jun. 2012.

[40] S. Eyerman and L. Eeckhout. A memory-level parallelism aware fetch policy for SMT processors. In *HPCA*, Feb. 2007.

[41] J. Friedman and N. Fisher. Bump hunting in high-dimensional data. In *Statistics and Computing*, 9, 1999.

[42] X. Fu, T. Li, and J. Fortes. NBTI tolerant microarchitecture design in the presence of process variations. In *Proceedings of the International Symposium on Microarchitecture (MICRO)*, Nov. 2008.

[43] M. Goraczko et al. Energy-optimal software partitioning in heterogeneous multiprocessor embedded systems. In *DAC*, Jun. 2008.

[44] N. Goswami, B. Cao, and T. Li. Power-performance co-optimization of throughput core architecture using resistive memory. In *HPCA*, Feb. 2013.

[45] R. Grant, and A. Afsahi. Power-performance Efficiency of Asymmetric Multiprocessors for Multi-threaded Scientific Applications. In *the 2nd work-shop on High-Performance, Power-Aware Computing, with IPDPS*, Apr. 2006.

[46] G. Groeseneken, F. Crupi, A. Shickova, S. Thijs, D. Linten, B. Kaczer, N. Collaert, and M. Jurczak. Reliability issues in MUGFET nanodevices. In *IEEE 46th Annual International Reliability Physics Symposium (IRPS)*, April 2008.

[47] E. Gunadi, A. A. Sinkar, N. S. Kim, and M. H. Lipasti. Combating aging with the colt duty cycle equalizer. In *Proceedings of 43rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Dec. 2010.

[48] V. Gupta et al. Using heterogeneous cores to provide a high dynamic power range on over-provisioned processors. In *Dark Silicon Workshop in conjunction with ISCA*, Jun. 2012.

[49] S. Hao, Q. Liu, L. Zhang, and J. Wang. Process Scheduling on Heterogeneous Multi-core Architecture with Hardware Support. In *NAS*, Jun. 2011.

[50] N. Hardavellas, M. Ferdman, B. Falsafi, A. Ailamaki. Toward dark silicon in servers. In *IEEE Computer Society*, 2011.

[51] T. Heath, B. Diniz, E. V. Carrera, W. Meria Jr., and R. Bianchini. Energy Conservation in Heterogeneous Server Clusters. In *PPoPP*. Jun. 2005.

[52] J. Hennessy, D. A. Patterson. Computer architecture: a quantitative approach. 5th edition.

[53] W. Huang, K. Rajamani, M.R.Stan, and K. Skadron. Scaling with design constraints - predicting the future of big chips. In *IEEE Computer Society*, 2011.

[54] R. Jammy. Materials, process and integration options for emerging technologies. SEMATECH/ISMI symposium, 2009.

[55] A. B. Kahng. The ITRS design technology and system drivers roadmap: process and status. In *Proceedings of 50th Design Automation Conference (DAC)*, Jun. 2013.

[56] P. L-Kamran et al. Scale-out processors. In *ISCA*, Jun. 2012.

[57] S. Kaxiras, and M. Martonosi. Computer Architecture Techniques for Power Efficiency. Morgan and Claypool Publishers.

[58] H. Kim, R. Vuduc, S. Baghsorkhi, J. Choi, and W. Hu. Performance analysis and tuning for general purpose graphics processing units (GPGPU). DOI:10.2200/S00451ED1V01Y201209CAC020.

[59] V. B. Kleeberger, H. Graeb, and U. Schlichtmann. Predicting future product performance: modeling and evaluation of standard cells in FinFET technologies. In *DAC*, Jun. 2013.

[60] O. Kocberber, B. Falsafi, K. Lim, P. Ranganathan, and S. Harizopoulos. Dark silicon accelerators for database indexing. In *Dark Silicon Workshop in conjunction with ISCA*, Jun. 2012.

[61] D. Koufaty, D. Reddy, and S. Hahn. Bias scheduling in heterogeneous multi-core architectures. In *EuroSys*, Apr. 2010.

[62] E. Kultursay, J, Swaminathan, V. Saripalli, V. Narayanan, M. Kandemir, and S. Datta. Performance enhancement under power constraints using heterogeneous CMOS-TFET multicores. In *CODES+ISSS*, Oct. 12.

[63] R. Kumar, K. I. Farkas, N. P. Jouppi, P. Ranganathan, D.M. Tullsen. Single-ISA Heterogeneous Multi-Core Architectures: The Potential for Processor Power Reduction. In *MICRO*. Dec. 2003.

[64] R. Kumar, D. M. Tullsen and N. P. Jouppi. Core architecture optimization for heterogeneous chip multiprocessors. In *PACT*, Sep. 2006.

[65] R. Kumar, D. M. Tullsen, P. Ranganathan, N. P. Jouppi, K. I. Farkas. Single-ISA Heterogeneous Multi-Core Architectures for Multithreaded Workload Performance. In *ISCA*. Jun. 2004.

[66] N. B. Lakshminarayna, J. Lee, H. Kim. Age based scheduling for asymmetric multiprocessors. In *SC*, Nov. 2009.

[67] H. Lee, C-H. Lee, D. Park, and Y-K. Choi. A study of negative-bias temperature instability of SOI and body-tied FinFETs. In *IEEE Electron Device Letters*, vol. 26, no.5, May 2005.

[68] J. Leng, T. Hetherington, A. Eltantawy, S. Gilani, N. S. Kim, T. M. Aamodt, V. J. Reddi. GPUWattch: enabling energy optimizations in GPGPUs. In *Proceedings of International Symposium on Computer Architecture (ISCA)*, Jun. 2013.

[69] S. Li et al. McPAT: an integrated power, area, and timing modeling framework for multicore and manycore architectures. In *MICRO*, Dec. 2009.

[70] T. Li, D. Baumberger, D. A. Koufaty, and S. Hahn. Efficient Operating System Scheduling for Performance-Asymmetric Multi-Core Architectures. In *SC*. Nov. 2007.

[71] T. Li, P. Brett, R. Knauerhase, D. Koufaty, D. Reddy, and S. Hahn. Operating System Support for Overlapping-ISA Heterogeneous Multi-core Architectures. In *HPCA*. Jan. 2010.

[72] C. Li, C. Ding, and K. Shen. Quantifying the cost of context switch. In *ExpCS*, Jun. 2007.

[73] Y. Li, B. C. Lee, D. Brooks, Z. Hu and K. Skadron. CMP design space exploration subject to physical constraints. In *HPCA*, 2006.

[74] M-L. Li, R. Sasanka, S. V. Adve, Y-K. Chen and E. Debes. The ALPBench benchmark suite for multimedia applications. UIUC CS Technical Report. Jul. 2005.

[75] H. Luo, Y. Wang, K. He, R. Luo, H. Yang and Y. Xie, Modeling of PMOS NBTI Effect of Considering Temperature Variation, In *Proceedings of International Symposium on Quality Electronics Design (ISQED)*, 2007.

[76] S. Mahapatra, P. B. Kumar, and M. A. Alam. Investigation and modeling of interface and bulk trap generation during negative bias temperature instability of p-MOSFETs. In *IEEE Transactions on Electron Devices*, vol. 51, no.9, Sept. 2004.

[77] M. Monchiero, P. Canal, and A. Gonzalez. Design space exploration for multicore architectures: a power/performance/thermal view. In *ICS*, Jun. 2006.

[78] C. H. Moore and G. Bailey. Illuminating dark silicon with a fabric of simple computers. In *Dark Silicon Workshop in conjunction with ISCA*, Jun. 2012.

[79] R. Nath, R. Ayoub, and T. S. Rosing. Temperature aware thread block scheduling in GPGPUs. In *Proceedings of 50th Design Automation Conference (DAC)*, Jun. 2013.

[80] T. Oh, H. Lee, K. Lee and S. Cho. An analytical model to study optimal area breakdown between cores and caches in a chip multiprocessor. In 2009 *IEEE Computer Society Symposium*.

[81] J. M. Rabaey, A. Chandrakasan and B. Nikolic. Digital Integrated Circuits, 2nd edition.

[82] P. Radojkovic et al. Optimal Task Assignment in multithreaded processors: a statistical approach. In *ASPLOS*, Mar. 2012.

[83] A. Raghavan et al. Computational Sprinting. In *HPCA*, Feb. 2012.

[84] A. Rahimi, L. Benini, R. K. Gupta. Aging-aware compiler-directed VLIW assignment for GPGPU architectures. In *Proceedings of 50th Design Automation Conference (DAC)*, Jun. 2013.

[85] K. Ramakrishnan, S. Suresh, N. Vijaykrishnan, M. J. Irwin, and V. Degalahal. Impact of NBTI on FPGAs. In *International Conference VLSI Design*, Jan. 2007.

[86] J. Renau et al. SESC Simulator.

[87] E. Saad, M. Awadalla, M. Shalan, and A. Elewi. Energy-aware task partitioning on heterogeneous multiprocessor platforms. In *IJCSI*, Vol. 9, 2012.

[88] J. C. Saez, A. Fedorova, M. Prieto, and S. Blagodurov. A comprehensive scheduler for asymmetric multicore systems. In *EuroSys*, Apr. 2010.

[89] J. C. Saez, A. Fedorova, D. Koufaty, and M. Prieto. Leveraging core specialization via OS scheduling to improve performance on asymmetric multicore systems. In *ACM Transactions on Computer Systems*, Apr. 2012.

[90] J. C. Saez, D. Shelepov, A. Fedorova, and M. Prieto. Leveraging workload diversity through OS scheduling to maximize performance on single-ISA heterogeneous multicore systems. In *Journal of parallel and distributed computing*, Aug. 2010.

[91] T. Sakurai and R. Newton. Alpha-power law MOSFET model and its applications to CMOS inverter delay and other formulas. In *IEEE Journal of Solid-State Circuits*, 1990.

[92] V. Saripalli, G. Sun, A. Mishra, Y. Xie, S. Datta, and V. Narayanan. Exploiting heterogeneity for energy efficiency in chip multiprocessors. In *IEEE Transactions on Emerging and Selected topics in Circuits and Systems*, Jun. 2011.

[93] V. Saripalli, A. K. Mishra, S. Datta, and V. Narayanan. An energy-efficient heterogeneous CMP based on hybrid TFET-CMOS cores. In *DAC*, Jun. 2011.

[94] S. Sharifi, A. K. Coskun, and T. S. Rosing. Hybrid dynamic energy and thermal management in heterogeneous embedded multiprocessor SoCs. In *ASPDAC*, Jan. 2010.

[95] J. Shin, V. Zyuban, P. Bose, and T. M. Pinkston. A proactive wearout recovery approach for exploiting microarchitectural redundancy to extend cache SRAM lifetime. In *International Symposium on Computer Architecture (ISCA)*, Jun. 2008.

[96] K. Singh, M. Bhadauria, and S. A. Mckee. Prediction-based power estimation and scheduling for CMPs. In *ICS* (extended abstract and poster presentation), Jun. 2009.

[97] S. Sinha, G. Yeric, V. Chandra, B. Cline, and Y. Cao. Exploring sub-20nm FinFET design with predictive technology models. In *Proceedings of 49th Design Automation Conference (DAC)*, Jun. 2012.

[98] A.Snavely, and D. M. Tullsen. Symbiotic Job Scheduling for a Simultaneous Multithreading Processor. In *ASPLOS*, Nov. 2000.

[99] W. Song, S. Mukhopadhyay, and S. Yalamanchili. Reliability implications of power and thermal constrained operations in asymmetric multicore processors. In *Dark Silicon Workshop in conjunction with ISCA*, Jun. 2012.

[100] S. Srinivasan, R. Iyer, L. Zhao, and R. Illikkal. HeteroScouts: Hardware Assist for OS Scheduling in Heterogeneous CMPs. In Poster session of the *ACM SIGMETRICS*. Jun. 2011.

[101] B. Swahn and S. Hassoun. Gate sizing: FinFETs vs 32nm Bulk MOSFETS. In *Proceeding of 43rd Design Automation Conference (DAC)*, Jun., San Francisco, CA, Jun. 2006.

[102] K. Swaminathan, E. Kultursay, V. Saripalli, V. Narayanan, M. Kandemir, and S. Datta. Improving energy efficiency of multi-threaded applications using heterogeneous CMOS-TFET multi-cores. In *Proceedings of International Symposium on Low Power Electronics Design (ISLPED)*, Sept. 2011.

[103] S. Swanson and M.B.Taylor. GreenDroid: exploring the next evolution in smartphone application processors. In *IEEE Communications Magazine*, Apr. 2011.

[104] S. Swanson et al. Area-performance trade-offs in tiled dataflow architectures. In *ISCA*, 2006.

[105] M.B.Taylor. Is dark silicon useful? In the *Proceedings of 49th Annual ACM Design Automation Conference* (*DAC)*, Jun. 2012.

[106] A. Tiwari and J. Torrellas. Facelift: Hiding and slowing down aging in multicores. In *Proceedings of the 41st IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Nov. 2008.

[107] G. Venkatesh, J Sampson, N. Goulding, S. Garcia. Conservation cores: reducing the energy of mature computations. In *ASPLOS*, Mar. 2010.

[108] G. Venkatesh et al. QSCores: Trading dark silicon for scalable energy efficiency with quasi-specific cores. In *MICRO*, Dec. 2011.

[109] F. Wang, Y. Xie, K. Bernstein, and Y. Luo. Dependability analysis of nano-scale FinFET circuits. In *Proceedings of the 2006 Emerging VLSI Technologies and Architectures (ISVLSI)*, Mar. 2006.

[110] L. Wang, K. Skadron, and B. H. Calhoun. Dark vs. Dim silicon and near-threshold computing. In *Dark Silicon Workshop in conjunction with ISCA*, Jun. 20012.

[111] W. Wang, V. Reddy and A. Krishnan, "Compact Modeling and Simulation of Circuit Reliability for 65-nm CMOS Technology", In *IEEE Transactions on Device and Materials and Reliability*, 7(4):509-517, Dec. 2007.

[112] Y. Wang, S.D. Cotofana, and L. Fang. Statistical reliability analysis of NBTI impact on FinFET SRAMs and mitigation technique using independent-gate devices. In *Proceedings of the IEEE/ACM International Symposium on Nanoscale Architecture (NANOARCH)*, Jul. 2012.

[113] X. Wu, J. Li, L. Zhang, E. Speight, R. Rajamony, and Y. Xie. Hybrid cache architecture with disparate memory technologies. In *ISCA*, Jun. 2009.

[114] G. Yan, Y. Li, Y. Han, X. Li, M. Guo and X. Liang. Agile regulator: a hybrid voltage regulator scheme redeeming dark silicon for power efficiency in a multicore architecture. In *HPCA*, Feb. 2012.

[115] J. Zhao, X. Dong and Y. Xie. Cost-aware three-dimensional (3D) many-core multiprocessor design. In *DAC*, Jun. 2010.

[116] L. Zhao, R. Iyer. S. Makineni, J. Moses, R. Illikkal, and D. Nowell. Performance, area and bandwidth implications on large-scale CMP cache design. In *HPCA*, 2007.

APPENDIX A. PERMISSIONS TO USE COPYRIGHTED MATERIALS

from permissions@hq.acm.org

reply-to Ying Zhang < yzhan29@tigers.lsu.edu>

date Fri, Oct 18, 2013 at 4:22 PM

subject Re: copyright questions in my dissertation

mailed-by hp.acm.org

Hello,

Thank you for your interest in ACM publications. Please forgive the delayed reply and confirm whether you received the attached auto-reply message directing you to use the RightsLink(r) online permission system in the ACM Digital Library to obtain a license for your request. As an author, the license allows you to include the published version of record, rather than the accepted version of your paper allowed under ACM Copyright Policy. www.acm.org/publications/policies/copyright_policy#Retained.

I hope this helps.

Regards,

Deborah Cotton

Copyright & Permissions

ACM Publications

212.626.0652

cotton@hq.acm.org

Dear Sir or Madam,

My name is Ying Zhang, and I am completing my PhD dissertation titled "Exploiting Heterogeneity in Chip-Multiprocessor Design" at Louisiana State University. Our school requires written permission from the publisher to use a published material in the thesis/dissertation, so I am writing to obtain your permission to reprint one of my papers appeared in the proceedings of the 50th Design Automation Conference (DAC'13) as one chapter in my dissertation. The detailed information about this paper is the following:

Y. Zhang, L. Peng, X. Fu, and Y. Hu, "Lighting the dark silicon by exploiting heterogeneity on future processors".

Please let me know if you have any questions. Your prompt response would be very appreciated! Thanks!

Regards,
Ying

ASSOCIATION FOR COMPUTING MACHINERY, INC. LICENSE

TERMS AND CONDITIONS

This is a License Agreement between Ying Zhang ("You") and Association for Computing Machinery, Inc. ("Association for Computing Machinery, Inc.") provided by Copyright Clearance Center ("CCC"). The license consists of your order details, the terms and conditions provided by Association for Computing Machinery, Inc., and the payment terms and conditions.

License Number	3259491261454
License date	Oct 31, 2013
Licensed content publisher	Association for Computing Machinery, Inc.
Licensed content publication	Proceedings of the 50th Annual Design Automation Conference
Licensed content title	Lighting the dark silicon by exploiting heterogeneity on future processors
Licensed content author	Ying Zhang, et al
Licensed content date	May 29, 2013
Type of Use	Thesis/Dissertation
Requestor type	Author of this ACM article
Is reuse in the author's own new work?	Yes
Format	Electronic
Portion	Full article
Will you be translating?	No
Order reference number	
Title of your thesis/dissertation	EXPLOITING HETEROGENEITY IN CHIP-MULTIPROCESSOR DESIGN
Expected completion date	Nov 2013

Estimated size (pages)	120
Billing Type	Credit Card
Credit card info	Visa ending in 9615
Credit card expiration	10/2016
Total	8.00 USD
Terms and Conditions	

Rightslink Terms and Conditions for ACM Material

1. The publisher of this copyrighted material is Association for Computing Machinery, Inc. (ACM). By clicking "accept" in connection with completing this licensing transaction, you agree that the following terms and conditions apply to this transaction (along with the Billing and Payment terms and conditions established by Copyright Clearance Center, Inc. ("CCC"), at the time that you opened your Rightslink account and that are available at any time at).

2. ACM reserves all rights not specifically granted in the combination of (i) the license details provided by you and accepted in the course of this licensing transaction, (ii) these terms and conditions and (iii) CCC's Billing and Payment terms and conditions.

3. ACM hereby grants to licensee a non-exclusive license to use or republish this ACM-copyrighted material* in secondary works (especially for commercial distribution) with the stipulation that consent of the lead author has been obtained independently. Unless otherwise stipulated in a license, grants are for one-time use in a single edition of the work, only with a maximum distribution equal to the number that you identified in the licensing process. Any additional form of republication must be specified according to the terms included at the time of licensing.

*Please note that ACM cannot grant republication or distribution licenses for embedded third-party material. You must confirm the ownership of figures, drawings and artwork prior to use.

4. Any form of republication or redistribution must be used within 180 days from the date stated on the license and any electronic posting is limited to a period of six months unless an extended term is selected during the licensing process. Separate subsidiary and subsequent republication licenses must be purchased to redistribute copyrighted material on an extranet. These licenses may be exercised anywhere in the world.

5. Licensee may not alter or modify the material in any manner (except that you may use, within the scope of the license granted, one or more excerpts from the copyrighted material, provided that the process of excerpting does not alter the meaning of the material or in any way reflect negatively on the publisher or any writer of the material).

6. Licensee must include the following copyright and permission notice in connection with any reproduction of the licensed material: "[Citation] © YEAR Association for Computing Machinery, Inc. Reprinted by permission." Include the article DOI as a link to the definitive version in the ACM

Digital Library. Example: Charles, L. "How to Improve Digital Rights Management," Communications of the ACM, Vol. 51:12, © 2008 ACM, Inc. http://doi.acm.org/10.1145/nnnnnn.nnnnnn (where nnnnnn.nnnnnn is replaced by the actual number).

7. Translation of the material in any language requires an explicit license identified during the licensing process. Due to the error-prone nature of language translations, Licensee must include the following copyright and permission notice and disclaimer in connection with any reproduction of the licensed material in translation: "This translation is a derivative of ACM-copyrighted material. ACM did not prepare this translation and does not guarantee that it is an accurate copy of the originally published work. The original intellectual property contained in this work remains the property of ACM."

8. You may exercise the rights licensed immediately upon issuance of the license at the end of the licensing transaction, provided that you have disclosed complete and accurate details of your proposed use. No license is finally effective unless and until full payment is received from you (either by CCC or ACM) as provided in CCC's Billing and Payment terms and conditions.

9. If full payment is not received within 90 days from the grant of license transaction, then any license preliminarily granted shall be deemed automatically revoked and shall be void as if never granted. Further, in the event that you breach any of these terms and conditions or any of CCC's Billing and Payment terms and conditions, the license is automatically revoked and shall be void as if never granted.

10. Use of materials as described in a revoked license, as well as any use of the materials beyond the scope of an unrevoked license, may constitute copyright infringement and publisher reserves the right to take any and all action to protect its copyright in the materials.

11. ACM makes no representations or warranties with respect to the licensed material and adopts on its own behalf the limitations and disclaimers established by CCC on its behalf in its Billing and Payment terms and conditions for this licensing transaction.

12. You hereby indemnify and agree to hold harmless ACM and CCC, and their respective officers, directors, employees and agents, from and against any and all claims arising out of your use of the licensed material other than as specifically authorized pursuant to this license.

13. This license is personal to the requestor and may not be sublicensed, assigned, or transferred by you to any other person without publisher's written permission.

14. This license may not be amended except in a writing signed by both parties (or, in the case of ACM, by CCC on its behalf).

15. ACM hereby objects to any terms contained in any purchase order, acknowledgment, check

endorsement or other writing prepared by you, which terms are inconsistent with these terms and conditions or CCC's Billing and Payment terms and conditions. These terms and conditions, together with CCC's Billing and Payment terms and conditions (which are incorporated herein), comprise the entire agreement between you and ACM (and CCC) concerning this licensing transaction. In the event of any conflict between your obligations established by these terms and conditions and those established by CCC's Billing and Payment terms and conditions, these terms and conditions shall control.

16. This license transaction shall be governed by and construed in accordance with the laws of New York State. You hereby agree to submit to the jurisdiction of the federal and state courts located in New York for purposes of resolving any disputes that may arise in connection with this licensing transaction.

17. There are additional terms and conditions, established by Copyright Clearance Center, Inc. ("CCC") as the administrator of this licensing service that relate to billing and payment for licenses provided through this service. Those terms and conditions apply to each transaction as if they were restated here. As a user of this service, you agreed to those terms and conditions at the time that you established your account, and you may see them again at any time at http://myaccount.copyright.com

18. Thesis/Dissertation: This type of use requires only the minimum administrative fee. It is not a fee for permission. Further reuse of ACM content, by ProQuest/UMI or other document delivery providers, or in republication requires a separate permission license and fee. Commercial resellers of your dissertation containing this article must acquire a separate license.

Special Terms:

If you would like to pay for this license now, please remit this license along with your payment made payable to "COPYRIGHT CLEARANCE CENTER" otherwise you will be invoiced within 48 hours of the license date. Payment should be in the form of a check or money order referencing your account number and this invoice number RLNK501148798. Once you receive your invoice for this order, you may pay your invoice by credit card. Please follow instructions provided at that time.

Make Payment To:
Copyright Clearance Center
Dept 001
P.O. Box 843006
Boston, MA 02284-3006

For suggestions or comments regarding this order, contact RightsLink Customer Support: customercare@copyright.com or +1-877-622-5543 (toll free in the US) or +1-978-646-2777.

Gratis licenses (referencing $0 in the Total field) are free. Please retain this printable license for your reference. No payment is required.

APPENDIX B. AUTHOR'S PUBLICATIONS

- (Under Review) **Y. Zhang**, L. Zhao, R. Illikkal, R. Iyer, A. Herdrich, and L. Peng, "QoS management on heterogeneous architecture for multi-programmed, parallel and domain-specific applications," Submitted for Review, 2013.

- (Under Review) **Y. Zhang**, L. Duan, B. Li, L. Peng, and S. Sadagopan, "Energy efficient job scheduling in single-ISA heterogeneous chip-multiprocessors," Submitted for review, 2013.

- (Under Review) **Y. Zhang**, S. Chen, L. Peng, and S. Chen, "Mitigating NBTI degradation on GPUs through exploiting device heterogeneity," Submitted for review, 2013.

- **Y. Zhang,** L. Duan, B. Li, L. Peng, and X. Fu, "Design configuration selection for hard-error reliable processors via statistical rules," To appear in Journal on Microprocessors and Microsystems.

- **Y. Zhang**, L. Peng, X. Fu, and Y. Hu, "Lighting the Dark Silicon by Exploiting Heterogeneity on Future Processors," In Proceedings of the 50th Design Automation Conference (*DAC*), Austin, TX, Jun. 2013.

- **Y. Zhang,** L. Duan, B. Li, and L. Peng, "Optimal Microarchitectural Design Configuration Selection for Processor Hard-Error Reliability", In Proceedings of the 13th IEEE International Symposium on Quality Electronic Design (*ISQED*), Santa Clara, CA, Mar. 2012.

- **Y. Zhang**, L. Peng, B. Li, J. Peir, and J. Chen, "Architecture Comparisons between Nvidia and ATI GPUs: Computation Parallelism and Data Communications", In Proceedings of IEEE International Symposium on Workload Characterization (*IISWC*), Austin, TX, Nov. 2011.

- **Y. Zhang,** Y. Hu, B. Li, and L.Peng, "Performance and Power Analysis of ATI GPU: A Statistical Approach", In Proceedings of the 6th IEEE International Conference on Networking, Architecture, and Storage *(NAS)*, Dalian, China, Jul. 2011.

- J. Chen, B. Li, **Y. Zhang**, L. Peng, and J.-K. Peir, "Tree Structured Analysis on GPU Power Study," In Proceedings of the 29th IEEE International Conference on Computer Design (*ICCD*), Amherst, MA, Oct. 2011.

- J. Chen, B, Li, **Y. Zhang**, L. Peng, and J.-K. Peir, "Statistical GPU Power Analysis Using Tree-based Methods", In Workshop of the second IEEE Greencomputing Conference (*IGCC*), Orlando, FL, Jul. 2011.

- L. Duan, **Y. Zhang**, B. Li, and L. Peng, "Universal Rules Guided Design Parameter Selection for Soft Error Resilient Processors," in *ISPASS*, 2011.

- L. Duan, **Y. Zhang**, B. Li, and L. Peng, "Comprehensive and Efficient Design Parameter Selection for Soft Error Resilient Processors via Universal Rules," To appear in IEEE Transactions on Computers.

- **Y. Zhang**, L. Peng, W. Lu, L. Duan, and S. Rai, "Expediating IP Lookups with Reduced Power via TBM and SST Supernode Caching," in *Computer Communications*, vol 33(3), pp. 390-397, Feb. 2010.

VITA

Ying Zhang was born in 1984, in Huanggang, Hubei, China. He received his Bachelor of Engineering and Master of Engineering degrees in Electronics and Information Engineering from Huazhong University of Science and Technology, Wuhan, China, respectively in June 2006 and June 2008. Since then, he has been enrolled in the Department of Electrical and Computer Engineering at Louisiana State University, Baton Rouge, Louisiana, to pursue his doctorate degree. During this period, he passed his qualify exam in Fall 2009 and general exam in November 2012, respectively. He anticipates graduating with his Doctor of Philosophy degree in Fall 2013.

Ying's research areas broadly lie in the area of computer architecture with particular interests in energy/cost-efficient heterogeneous system design, GPU architecture and processor hard-error reliability. He has published a series of papers on these topics in various computer architecture conferences and journals. He also finished a research internship at Intel Research Labs, Hillsboro, Oregon, from December 2011 to June 2012, working on OpenCL performance characterization on Intel Sandy Bridge and Ivy Bridge platforms.