Title
Power-Aware Core Management Scheme for Heterogeneous Many-Core Architecture

Author
Kim, Myoungseo

Publication Date
2015

Peer reviewed|Thesis/dissertation

UNIVERSITY OF CALIFORNIA,
IRVINE

Power-Aware Core Management Scheme for Heterogeneous Many-Core Architecture

THESIS

submitted in partial satisfaction of the requirements
for the degree of

MASTER OF SCIENCE

in Computer Engineering

by

Myoung-Seo Kim

Thesis Committee:

Professor Jean-Luc Gaudiot, Chair

Professor Nader Bagherzadeh

Professor Alexandru Nicolau

2015

DEDICATION

To my father and mother,

Youngkyu Kim and Heesook Park

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

First of all, I would like to thank and praise God to give me wisdom, knowledge, and strength, that I make all these possible against every temptation and adversity. I am also deeply respectful and grateful to my advisor, Dr. Jean-Luc Gaudiot, for his encouragement, guidance and patience during my study. I was very fortunate to meet him as an advisor. In addition, I have learned many aspects of computer science and engineering from his incredible and creative insight and been inspired by his passion for work.

I would also like to say thank you to my committee members: Professor Nader Bagherzadeh for his trust; and Professor Alexandru Nicolau for his kindness support and encouragement. I wish to express best regards and blessing to all my colleagues in parallel systems & computer architecture lab (PASCAL).

Additional support is provided by the National Science Foundation under Grant No. CCF-1065448. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

Special thanks to Korean graduate student members for being my great supporter and for helping me on everything to be enriched the life in Irvine.

Finally, I give my sincerest gratitude and honor to my family who have patiently supported and prayed for me to move forward and to achieve my dream finally.

ABSTRACT OF THE THESIS

Power-Aware Core Management Scheme for Heterogeneous Many-Core Architecture

By

Myoung-Seo Kim

Master of Science in Computer Engineering

University of California, Irvine, 2015

Professor Jean-Luc Gaudiot, Chair

The main challenge in designing the future heterogeneous many-core architecture on the same chip is to provide a solution that has low power consumption, in addition to trade off a small decrease in performance and throughput. Our design incorporates heterogeneous cores representing different points in the power-performance design space during an applications execution. Under this circumstance, system software dynamically chooses the most appropriate core to meet specific performance and power requirements. In this paper, the authors present a power-aware core management scheme based on efficient control of the core resources on heterogeneous many-core architecture as a mechanism to reduce a huge latency and a power dissipation for powering the core up from powered down. Operation is based on distinct scenarios by 3-bit core power control scheme through 5 statuses switching such as active, hot core, cold core, idle, and powered down. In addition, for more elaborated control to be power-performance efficient, this kind of status switching is exactly triggered by power-aware thread placement through heuristic thread consolidation approach. To achieve this objective, we have tried to deal with this scheme in terms of calculating peak and typical power consumption and managing core resources efficiently.

Index Terms - Power Management, Heterogeneous Many-Core Architecture, 3-bit Core Power Control Scheme, Power-Aware Thread Placement, Heuristic Thread Consolidation.

Chapter 1

Introduction

As processors continue to increase in performance and speed, processor power consumption and heat dissipation have become key challenges in the design of future heterogeneous many-core architecture of a dark silicon era. In addition, heterogeneous system architecture (HSA) foundation [1] and international technology roadmap for semiconductor (ITRS) organization [2] including top-tier industry researchers have advocated future many-core chips [3] over heterogeneous multicore systems such as AMD Fusion [4], Intel Haswell [5], Nvidia Denver [6], and ARM big.LITTLE [7], power consumption for future large-scale processors nevertheless remains a serious concern. Increased power consumption and heat dissipation typically leads to higher costs for thermal packaging, fans, electricity, and even air conditioning. Higher-power systems can also have a greater incidence of failures.

Prior chip-level many-core architecture has been proposed using multiple copies of the same core, namely homogeneous [8]. For many applications, core diversity is of higher value than uniformity, offering much greater ability to adapt to the demands of applications [9]. We present a heterogeneous many-core architecture where all cores execute the different instruction set, capabilities and performance levels.

As the increasing scale of cores on a chip, the more communications and data movements among cores will be occurred, and the whole computing performance would be decreased while the power dissipation would be increased sharply if there is not a good solution for it. All of the prior work inspired us to eliminate the overhead of power consumption. And, we figured out an efficient core power management method by designing a 3-bit control scheme in consideration of power-aware thread placement for real heterogeneous many-core architecture.

One of the motivations for this proposal is that typical programs go through phases with different execution characteristics [10], [11]. Thus, the best core during one execution flow may not fit into for the next execution flow. This observation motivates the ability to switch cores dynamically among execution flow. In addition, unused cores are completely powered down in this situation, rather than left idle. Therefore, unused core suffer no static leakage or dynamic switching power. This approach, however, has a huge latency penalty for powering a new core up. The latency penalty is in approximately one thousand cycles of 2GHz clock [9]. It is more severe in an increasing clock frequency under many cores near future.

In this paper, we propose an advanced core power management scheme for designing heterogeneous many-core architecture. The key idea is to process 3-bit core power control scheme through status switching based on power-aware thread placement. Using this approach, per-core can be controlled independently, thereby reducing processor power dissipation, while maintaining consistent performance possibly.

The remainder of this paper is organized as follows. After Chapter 2, which presents the related works, Chapter 3 introduces the target architecture. Then, detailed structure and functions of the proposed method are introduced in Chapter 4. Chapter 5 discusses about power-aware thread placement. Chapter 6 shows the evaluation and methodology based on our previous and current work, followed by future work in Chapter 7. Finally, Chapter 8 summarizes this work.

Chapter 2

Related Work

Previous work on power-related optimizations for processor design can be broadly classified into four categories:

(1) work that uses voltage and frequency scaling of the processor core to reduce power [12]. Dynamic voltage and frequency scaling (DVFS) is a popular method for power management due to the cubic dependency of dynamic power about frequency scales with voltage squared. To identify the best voltage-frequency (V-F) setting, the main stream of recent work on DVFS control relies on information gathered from performance counters during runtime. In this case, the need for special compiler support or modications to the applications is ignored [25], [26]. Some approaches for power control via software require application-level or compiler-level support [27], [28]. Most methods focus on optimizing metrics such as energy, energy-delay-product (EDP) and energy-delay-squared-product (ED^2P). The paper by Canturk Isci et al. [29] derives phase categories based on a metric for the memory operation rate (mem/μ-op), and each category is a mapped to an optimal V-F setting. Similarly, in the paper by Gaurav Dhiman et al. [30], propose an online learning model for single-core processors. In order to characterize workloads, they break down the cycles per instruction (CPI) metric

into various components such as baseline CPI, miss events CPI, and stall CPI. This approach guarantees convergence to the optimum V-F setting using online learning. These approaches focus on energy, EDP or ED^2P minimization without considering power caps.

Most multicore processors support independent frequencies control for the cores but a common voltage level is usually set to support the highest frequency. Independent voltages require extensive design investments in the power-delivery network and the off-chip power regulators. To increase the granularity of DVFS control, multiple clock domain design and voltage frequency island partitioning have been proposed [36], [37]. To reduce the overhead of runtime voltage conversion, the paper by Wonyoung Kim et al. [38] explores designing on-chip regulators and perform core-level DVFS.

(2) work that uses gating technique in terms of the ability to turn on and off portions of the core for power management [13]. Power gating is a circuit-level technique that allows to cut off the power supply to a logic macro. It is implemented with the help of a sleep transistor that is inserted as a series header or footer device in the V_{DD}-to-V_{SS} circuit path that includes the targeted macro. In particular, per-core power gating (PCPG) [55] is becoming an increasingly common knob in todays microprocessors [56], [57], [58]. Nevertheless, how to make the most proper use of PCPG is still an open question. For example, actuating PCPG every time a core becomes idle may lead to negative power-performance benefits if the core idleness period is not long enough. Evidently, processes and software threads have to be scheduled accordingly across cores to generate opportunities beneficial for PCPG, with minimal impact on performance.

(3) work that enables larger degrees of freedom in job scheduling and allocation. Rangan et al. propose a scalable DVFS scheme for multicore systems that enables thread migration among homogeneous cores with heterogeneous power-performance capabilities [31]. Rather than changing the V-F settings on demand, they assign fixed V-F settings to different cores and migrate the applications to reach the desired level of performance within a given power

budget. For applying DVFS under power constraints, Etinski et al. propose a job scheduling policy that optimizes performance for a given power budget [32]. Isci et al. evaluate global power management policies with objectives such as prioritization, power balancing and optimized throughput for various benchmark combinations and power budgets [34]. However, their approach does not provide dynamic adaptation to different workloads. Teodorescu et al. propose algorithms for power management through scheduling and DVFS under process variations [35].

(4) work that makes thread consolidation and motion based on thread migration betwwen cores with different voltage and frequency settings. The paper by Cochran et al. [59] is most closely related to this work. Also, Tam et al. [60] propose a mechanism for thread clustering based on data sharing patterns. It is implemented at operating system (OS) kernel level with information from hardware event counters. This work closely relates to ours in the methodology they use, which is also based on dynamic analysis of processor counters. However, they just tackle performance improvement while we also consider power reduction. Rangan et al. [61] present thread motion, a technique capable of fine-grained power management based on thread migration between cores with different V-F settings. The use of hardware event counters in the context of multi-threaded applications is also leveraged in prior studies. In addition to Tam et al.'s work, Bhattacharjee et al. [62] also propose the use of processor counters to dynamically predict thread criticality. A critical thread is the slowest thread in an application, which limits its performance. They propose to exploit thread criticality prediction for load balancing and energy saving purposes.

As the need for power capping and peak power management grows, some of the recently proposed techniques have explicitly focused on meeting power budgets or peak power constraints at runtime. Cebrian et al. propose a power balancing strategy that dynamically adapts the per-core power budgets depending on the workload characteristics [39]. However, for balanced workloads which have even power consumption among cores in the Parsec

benchmark suite [17], this strategy would not perform well as it relies on borrowing power budgets from cores that consume lower power than the others. Sartori et al. propose a peak power management technique for multicore systems by choosing the power state for each core that meets the power constraints [40].

The power capping strategy proposed by Gandhi et al. meets the power budget by inserting idle cycles during execution [41]. This approach targets controlling the average power consumption, and does not provide peak power guarantees. A number of approaches meet the power budgets through hardware reconfiguration. Meng et al. propose a power management strategy through dynamic reconfiguration of cores by cache resizing [42]. Kontorinis et al. propose a table-driven adaptive core reconfiguration technique that configures core resources such as floating point units and load-store queues to meet peak power constraints [43].

Current processor and system vendors have begun to provide peak power management features in commercial products. AMD has introduced PowerCap Manager for 45nm Opteron processors [44]. For data center power management, HP and Intel jointly offer a power capping technique which adjusts power caps according to busy/idle states of the nodes [24]. This technique utilizes the DVFS states and the throttling capabilities for idle cycle insertion at the chip-level. Besides sleep modes, power nap modes, in which the system can enter and exit from low-power modes in milliseconds, have been also proposed to cope with the demand variation patterns in data centers [45].

Our many-core heterogeneous architecture does not preclude the use of these techniques and can potentially address the drawbacks of these techniques to provide much greater power savings.

Chapter 3

Architecture

This Chapter gives an overview of a potential heterogeneous many-core architecture and discrete second level cache memory model.

3.1 Heterogeneous Many-Core System

In Fig. 3.1, the architecture consists of 3-level computing elements such as core, quart, and tile. The cluster is characterized by a group of cores containing CPU and GPU. The heterogeneity of core in such architecture can be made 4 CPU cores and 12 GPU cores sharing adaptive L2 cache among each type of cores. In default, each CPU and GPU core has its private L1 cache independently. Then, each cluster is a quarter of a tile, and each quarter shares their data by input and output queues over high throughput network on chip. Thereby, each tile has 16 CPU and 48 GPU cores. This composition is similar to the architecture which is introduced by [8], [15].

3.2 Discrete L2 Cache Memory Model

Growing core counts have highlighted the need for scalable on-chip cache coherence mechanisms. The increase in the number of on-chip cores exposes the power and area costs of scaling the directories. Therefore, for separate L2 cache for CPUs and GPUs cluster, we used a 4-way cuckoo directory [14] (see Fig. 3.2) for each CPUs and GPUs cluster in order to decrease the possibility of replaced block because of the high reusable frequency of interior data blocks. To find an element in the cuckoo directory, all ways are looked up in parallel using hashed values of the searched address. Inserting an entry into the directory requires a lookup followed by a write of an entry in one of the ways. If the write replaces a valid directory entry, the insertion procedure is repeated for the victim entry, iterating until an insertion finds a vacant location.

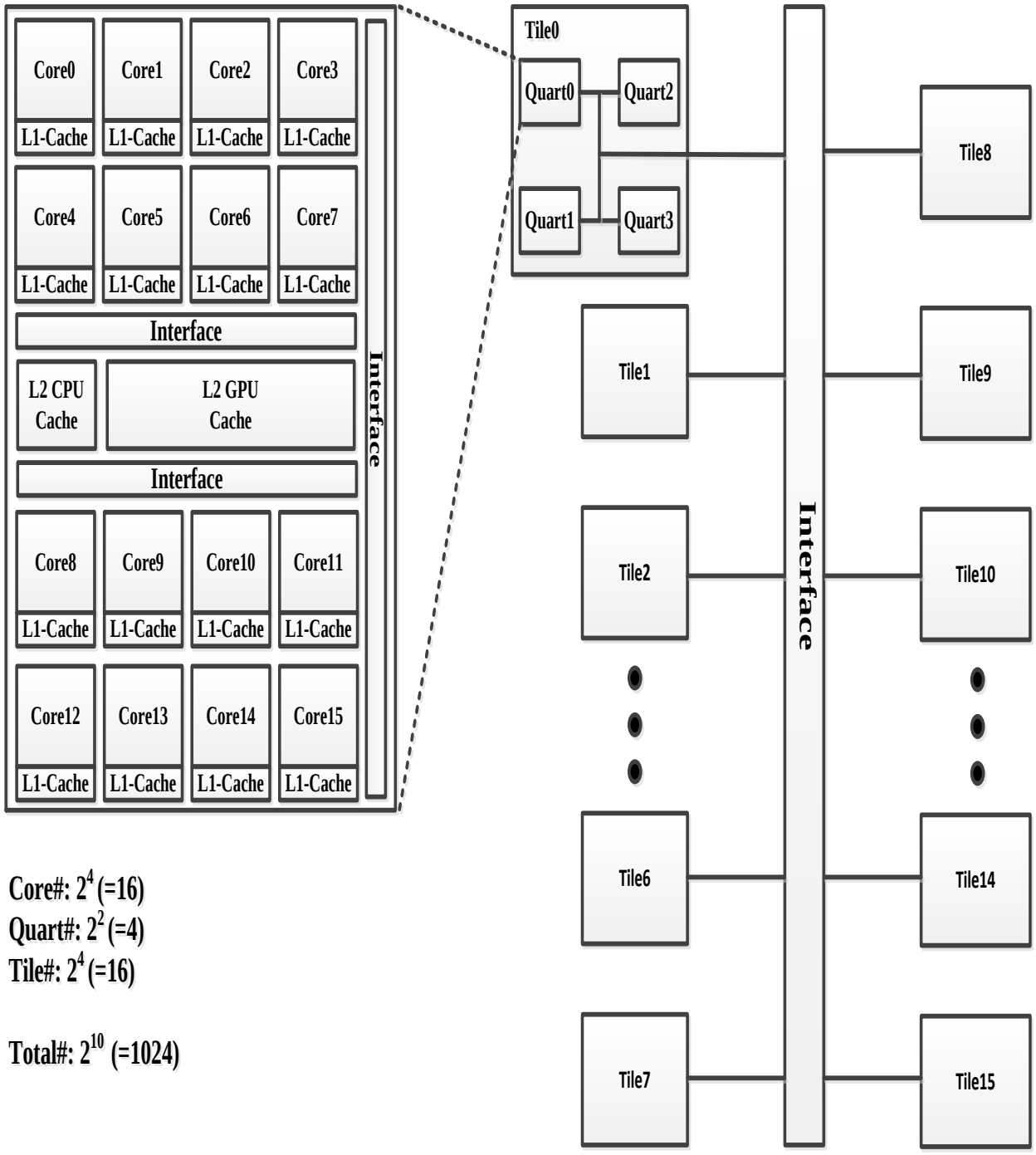


Figure 3.1: Heterogeneous Many-core architecture.

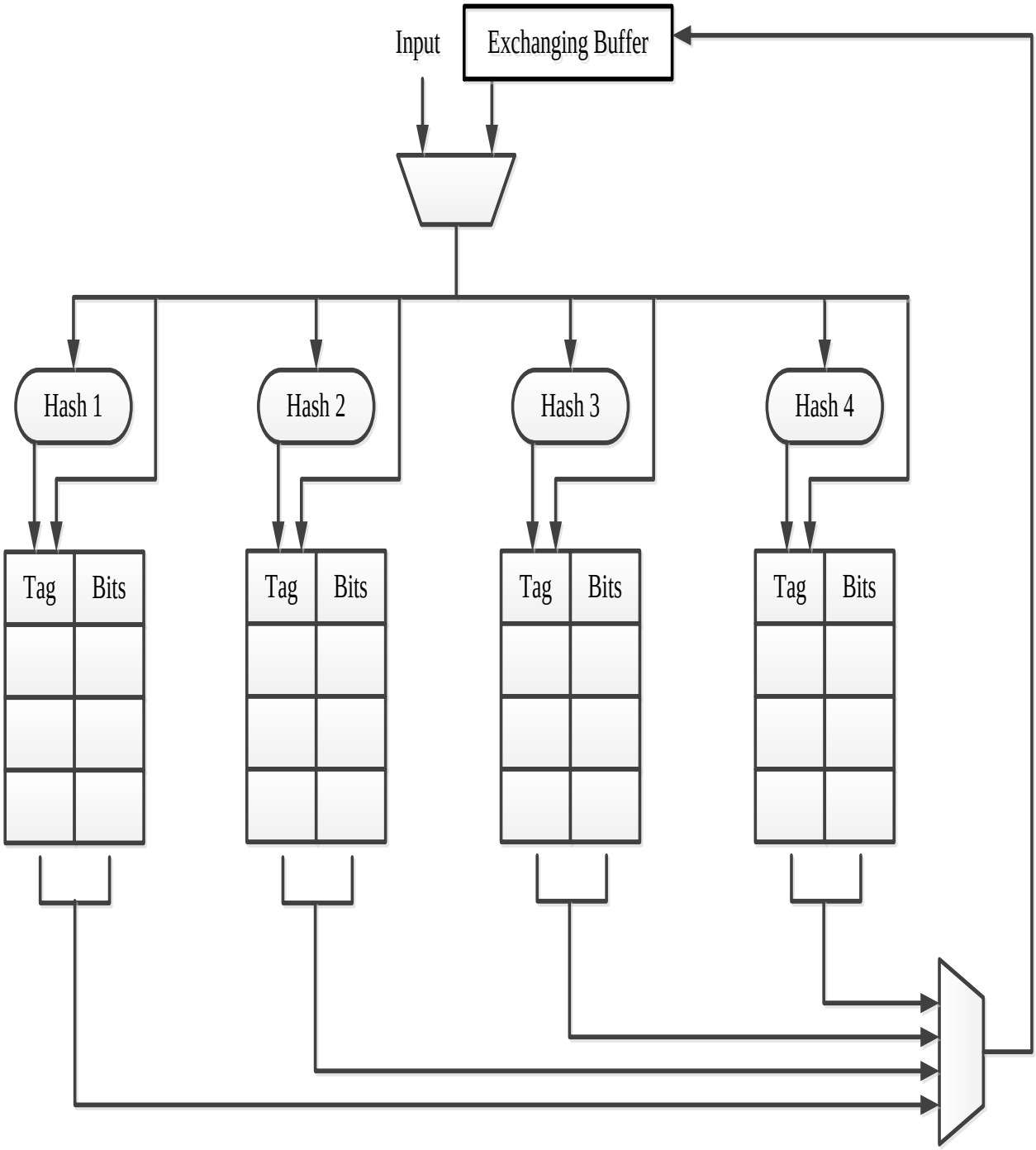


Figure 3.2: 4-way cuckoo directory structure.

Chapter 4

3-Bit Power Control Scheme

So far we have discussed about our heterogeneous architecture model using many of small cores. Now we discuss how to fit the heterogeneous many-core architecture in an affordable power envelope.

The best lever to reduce power with minimal impact on performance is to use voltage scaling. Applying voltage scaling indiscriminately to the entire architecture would lower power, but may not be optimal. Instead we propose to exploit the fact that there are many of cores, and each core can be 3-bit power controlled separately, therefore employing fine grain power management.

Individual cores can be voltage and frequency scaled to any arbitrary voltage and frequency in the possible range, but this could be cumbersome and hard. Also, different core running at different voltages and frequency would create asynchronous interface, additional latency, meta-stability, and would require complex power delivery scheme.

We propose a much simpler method of 3-bit core power management scheme in Fig. 4.1 and Fig. 4.2. As shown in the finite state machine (FSM) in Fig. 4.1, On/Off indicators around

arrows between statuses represent that it is triggered by the OS. A core operates at one of the five statuses: active, hot core, cold core, idle and powered down. The 3-bit core power management scheme always assigns a 3-bit counter to an each core when each application program switches. And, idle status must keep twice before it is changed to the powered down. The counter value is between 0 and 2^n-1, n is 3 bits in this case. The 3 bits are used to encode the 5 states in this architecture. When the counter is greater than or equal to one-half of its maximum value (2^n-1), the power status is either active or hot/cold core; otherwise, it is idle or powered down. In a counter operation, the counters are incremented when a status is On and decremented when a status is Off; the counters saturate at 000 or 100. In case of the dotted arrow located on the FSM in Fig. 4.1, idle status can be changed to powered down status to remove a static leakage when a transition status is continuously Off over 2 times iterations.

Fig. 4.3 shows the power and clock distribution used in the heterogeneous many-core architecture, assuming upper four cores (see red colored cores, namely hot cores) are all frequently used and lower four cores (see blue colored cores, namely cold cores) are all less frequently used. Also, two main contributions from peripherals can be observed: the upper line which goes to cores shows the power line and the lower line which goes to cores represents the clock line.

This scheme focuses on predictively transitioning the power status that some hot cores will change again soon from slow active status to active status. Thus, the strategy avoids the power dissipation due to the latency and associated leakage penalty incurred by accessing from an idle status. Since sequentiality among jobs running on core is the norm in execution flow in heterogeneous many-core architecture, the OS predictively decides on a power status of hot cores as the slow active for preparing the future usage. Also, hot cores can be exactly decided by the OS based on both the cores usability profiling information and real time sensing information getting from the temperature sensor of an each core. Hot spot on the

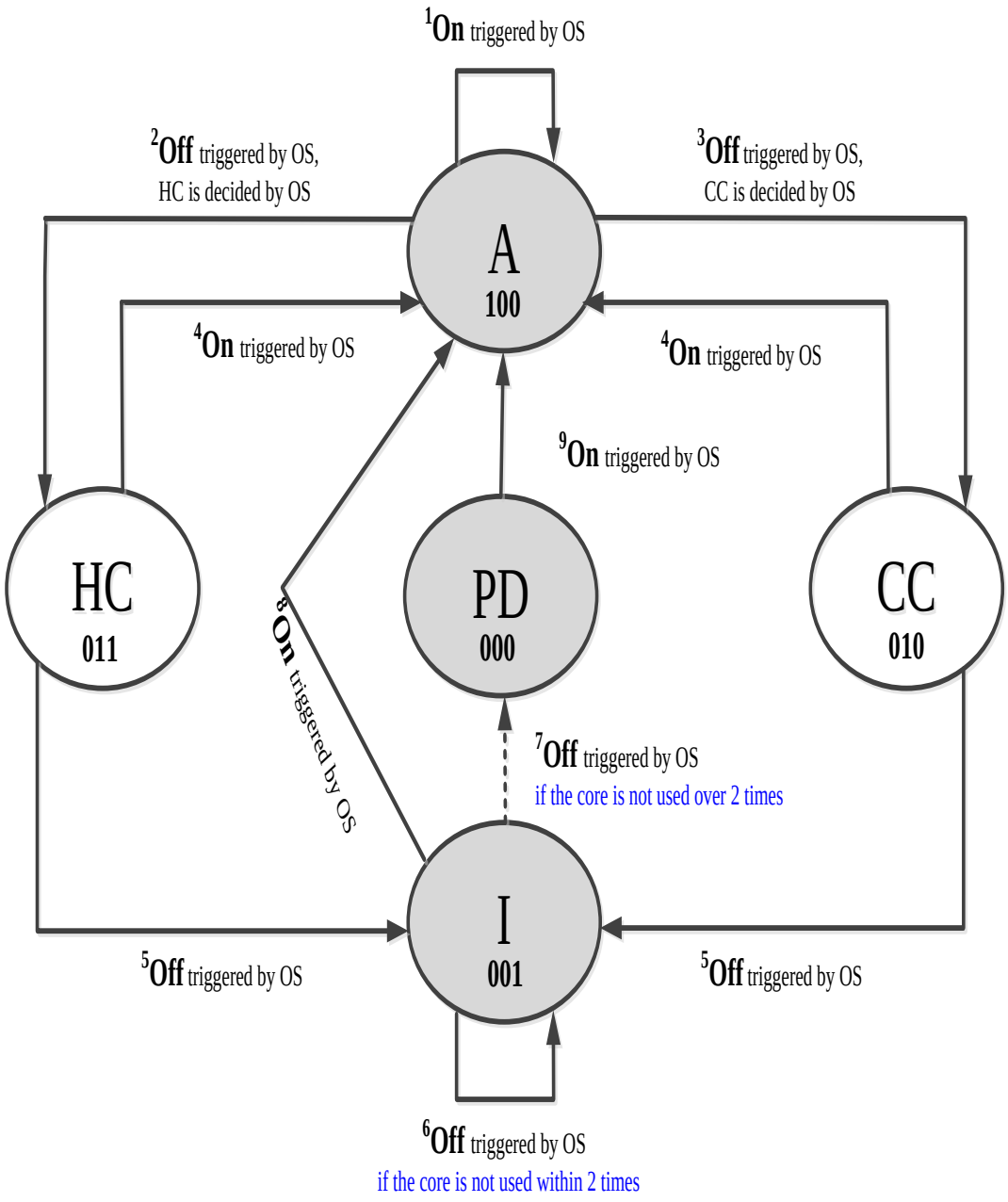
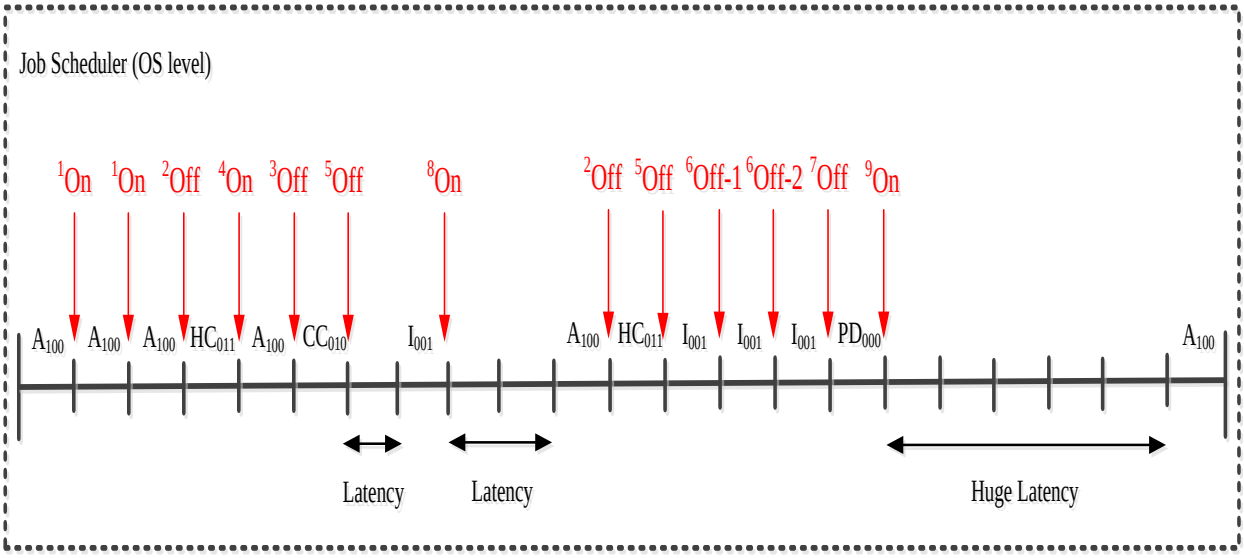


Figure 4.1: 3-bit core power control scheme under FSM.



A_{100}: Operational state. Core fully turned on.

HC_{011}: Keep Core main internal clock slow if it is the hot core. Bus interface unit is kept running at full speed. Also, the Core keep its cache coherent. It might be returned to A_{100} state in a few cycles.

CC_{010}: Stop Core main internal clock if it is the cold core, but the PLL is kept running. Bus interface unit is kept running at full speed. Also, the Core keep its cache coherent. It might be returned to A_{100} state in a few ten cycles.

I_{001}: Stops Core main internal clock and the PLL. State where the core maintains all software-visible states, but do not need to keep its cache coherent. It may take longer to wake up through interrupts (It might be spent 400 cycles in 2GHz).

PD_{000}: Core fully turned off. It may take over 1000 Cycles(in 2GHz) to wake up through interrupts.

Figure 4.2: 3-bit core power control scheme under the operating sequence.

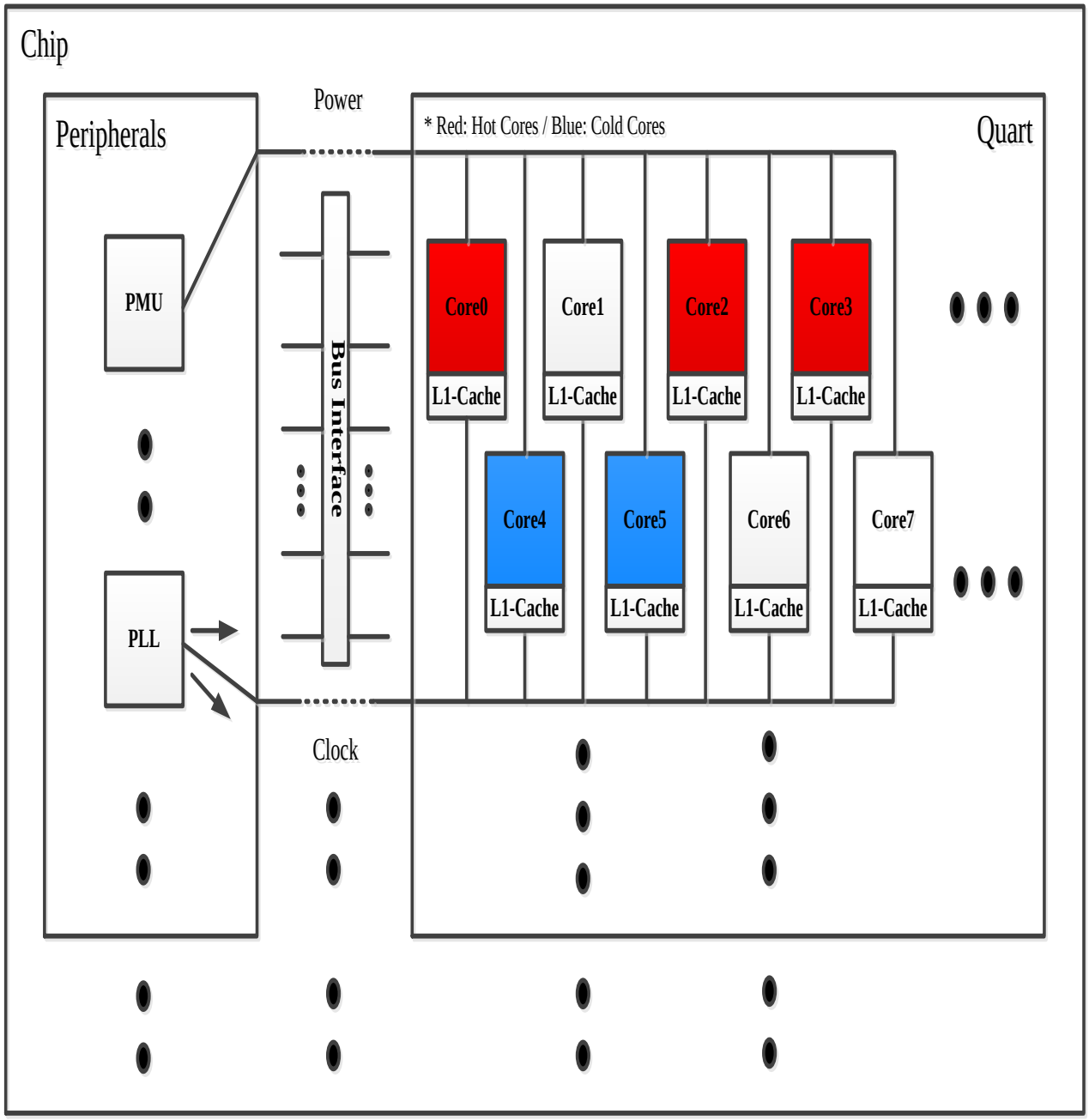


Figure 4.3: Power and clock distribution.

chip gives a clue to make a decision more clear with regard to hot cores.

Table 4.1 summarizes the various core status supported by 3-bit core power control scheme of the heterogeneous many-core architecture and various hardware that is related to our scheme. The following describes the various actions that are performed in the various core statuses supported by the 3-bit core power control scheme of our heterogeneous many-core architecture.

4.1 Active Status

When the core is in the active status, the core is also defined to be in status A as an acronym. In status A, instructions are actively be executed by a particular core or all cores, depending on the current job scheduling.

4.2 Hot Core Status

When the core is in the hot core status, the core is also defined to be in status HC as an acronym. If the core is decided to the hot core by OS, some instructions are being executed with slow clock to stay alive. Based on the cores usability profiling information and real time sensing information getting from the temperature sensor of an each core, we can predict that this core might be soon reused near the future.

4.3 Cold Core Status

When the core is in the cold core status, the core is also defined to be in status CC as an acronym. If the core is decided to the cold core by OS, no instructions are being executed. Also, the clocks of all clock trees pertaining to the cold cores are turned off. Although the clocks are off by utilizing a technique known as clock gating, the phase-locked loop (PLL) is still alive in this status. Therefore, we can significantly reduce the latency rather than changing from Idle status to Active status. The only power still consumed is caused by leakage current. However, it is compensated by the power saving according to the latency reduction.

4.4 Idle Status

When the core is in the idle status, the core is also defined to be in status I as an acronym. While in idle status, the core PLL is turned off, and the core cache is flushed. A core in this status is considered an inactive core. The wakeup time for this status is significantly longer than in slow active since the core cache must be restored. In addition, the PLL needs a time to be stabilized for generating the correct frequency.

4.5 Powered Down Status

When the core is in the powered down status, the core is also defined to be in status PD as an acronym. While in powered down status, the core PLL is turned off, and the core cache is also flushed. And, the core status is saved to the L3 cache, namely the last level cache (LLC) here. Additionally, the core is power gated to reduce power consumption to the core to approximately zero watts. A core in this status is considered an inactive core. The

Table 4.1: Processor Power Design Space

	Active	Hot[1] Core	Cold[2] Core	Idle	Powered Down
Core Clock	On (Full)	On (Slow)	Off	Off	Off
PLL	On	On	On	Off	Off
Core Power	On	On	On	On	Off
PMU	On	On	On	On	Off
Core Caches	Keep	Keep	Keep	Flushed	Flushed
Shared Cache	Keep	Keep	Keep	Keep	Keep
Wakeup Time	Active	Few Cycles	Few Ten Cycles	Few Hundred Cycles	Few Thousand Cycles

[1]Hot core: the frequently used core
[2]Cold core: the less frequently used core

wakeup time for this status is the longest. The core status has to be restored from the L3, the core PLL must be stabilized, the power gating must be deactivated, and core clock must be turned back on.

An interesting situation may occur in idle and powered down status. Since I and PD make the significant latency for powering up to A status in this architecture model, the energy cost to transition to and from these statuses is very high, particularly in PD status. Frequent transition in and out of these statuses can result in an extreme energy loss. To prevent this, our 3-bit core power control scheme is very useful to determine when SA status savings justify the energy cost of transitioning into I and PD status and then transition back to A. If there is not enough justification to transition to PD status, the power management unit (PMU) requests the OS to stay I status.

Table 4.2: Each Core Power Approximate Calculation

Fine Grain Models for an Each Core Status					
	Active	Hot Core	Cold Core	Idle	Powered Down
Active (Dynamic) CV^2F	Max_D	Max_D x 50%	N/A	N/A	None
Standby (Leakage) VI_{off}	N/A	N/A	Max_L	Max_L x 60-80%	None
Total[1]	Max_D	$\geq$ (Max_D x 50%)	Max_L	$\geq$ (Max_L x 60-80%)	None
Wakeup[2]	N/A	Few Cycles x 200W[3]	Few Ten Cycles x 200W[3]	Few Hundred Cycles x 200W[3]	Few Thousand Cycles x 200W[3]

[1]Total = Active + Standby

[2]Wakeup = Cycle count x Power per cycle[3]

[3]Power per cycle: Approximately 200Watts at 2GHz

Table 4.2 shows fine grain power models along with an each core status such as active, standby, total, and wakeup power.

We assumed that the power consumption per cycle is approximately 200W at 2GHz. Also, each job was characterized in the same operating conditions and during its execution no competing tasks were performed. Additionally, Equation (4.1) shows the full chip level power consumption by combining with the summation of all cores power consumption (i.e.,

$1 \leq$ i $\leq$ n , n is the total number of cores) and some peripherals power consumption.

$$P_{chip} = \sum_{i=1}^{n} P_{core(n)} + P_{peripherals} \tag{4.1}$$

Chapter 5

Power-Aware Thread Placement

This is a sub-part to elaborate our 3-bit core power control scheme. In Chapter 4, I told that status changing between various core statuses is triggered by OS. This type of status transition is made by core's On/Off singnal as shown in Fig. 4.1, namely the FSM of 3-bit core power control scheme. The meaning of "core status changing is triggered by OS" is exactly based on the power-aware thread placement. Thread placement is performed today in a largely power-aware manner. In particular, consolidation of active threads into fewer cores exposes opportunities for power savings. In this situation, power saving opportunity is especially high because PCPG is becoming viable. It is applicable to unused cores which is excluded from thread consolidation.

The incorporation of multiple threads and cores into the same chip affects the hardware-software interaction signicantly. The allocation of software threads across available hardware threads is a software-level responsibility, with little or no hardware control. Therefore, programmers rely on OS schedulers to allocate software threads across hardware threads. They can also explicitly establish preferred thread allocations by setting affinities between software and hardware threads.

The use of the optimum combination of core-wise simultaneous multi-threading (SMT) level and number of active cores needs to achieve a desired power-performance efficiency. Based on my survey, this kind of approach is not been explored in previous work nor implemented as part of the OS task scheduler. However, we have a limitation through this approach. Under multi-threaded workloads, the best thread allocation policy to improve performance is not always evident and how thread allocation might impact the power consumption is also not always evident. But, it is an inevitable trend for this research field. So, we are now trying to figure out an optimal solution.

In Fig. 5.1, conguration "4x2" allows us to switch the four cores that are left unused to idle state. Software threads are pinned to specic hardware threads by setting CPU afnities. Total HW threads (T) is to multiply the number of cores (C) by SMT threads per core (S). For instance, in this case, 16 is to multiply 8 by 2. We benet from unused cores to reduce power consumption (e.g. by switching them to an hot/cold core status or idle/powered-down status). In addition, each core can effectively access shared data located in remote L2 and L3 caches through the coherence fabric. Execution time is increased by just 5% when conguration "4x2" is adopted, chip power consumption is cut down by slightly more than 20% [52], [53]. The reason for such small performance degradation when the number of cores is halved relies on the signicant inter-thread data sharing present in this application. When software threads are executed closer (sharing the same cache hierarchy), the number of accesses to remote cache regions in the chip decreases signicantly. The idea of thread consolidation in SMT-enabled Chip Multiprocessors (CMP) is discussed in very few prior works. Among them, the most good work which I think is as follows [59]. It packs software threads onto a variable number of cores to t a given power budget, in conjunction with DVFS. The work examines different thread packing and DVFS congurations to maximize performance within variable power caps. However, it does not take into account the actual software-hardware thread mapping and SMT is deliberately disabled.

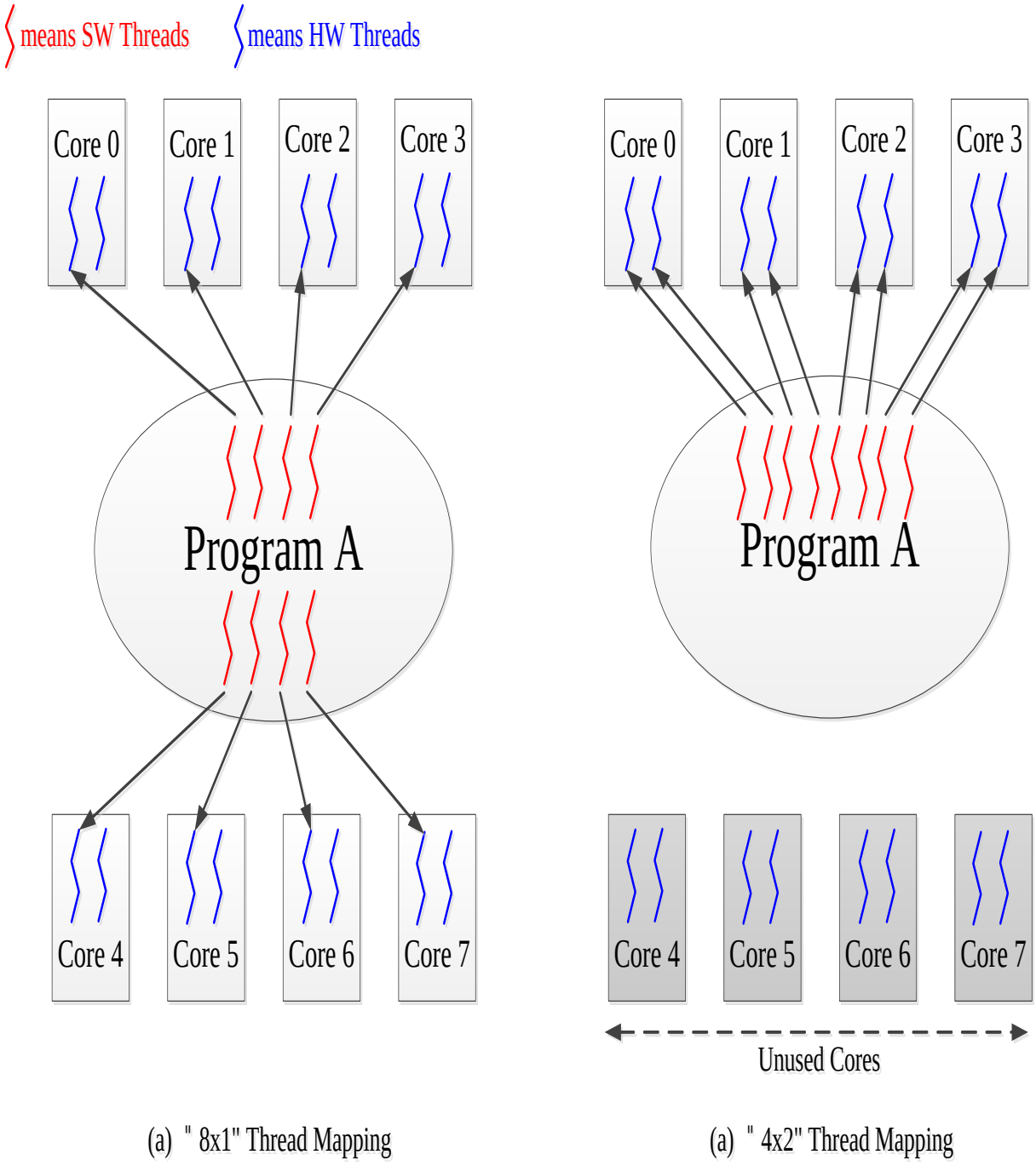


Figure 5.1: Hardware-Software Thread Interaction.

Multi-threaded applications may not benet from statically pinning software threads to fewer cores. It strongly depends on the applications characteristics during its different execution phases. Therefore, we need to find sweet spots dynamically which represent particular software thread placements to maximize power-performance efficiency. It is more convenient to use a thread consolidation which is the method of using fewer cores at higher SMT levels.

Our objective is to detect (during an applications execution) when it is possible to consolidate threads with minimal or zero impact on performance. Similarly, if threads placed in the same core face high inter-thread conicts , the heuristic unconsolidates them to mitigate the situation. To build and evaluate a heuristic capable of nding the most power-performance efcient thread mappings at runtime, we required to have a metric to quantify such efficiency. This is a efficiency of thread consolidation (E_TC) as shown in Equation (5.1). It is also used in [54]. If E_TC is larger than 1, the new new mapping (mapping A) provides larger power-performance efciency than the previous one (mapping B). Ideally, we are interested in actions (consolidations or unconsolidations) with E_TC values larger than one. If E_TC is smaller than 1, the new mapping is less power-performance efcient than the previous one. At last, if E_TC is equal to 1, both mappings perform equally in terms of power-performance efciency. For example, lets assume that consolidation reduces chip power consumption by 20% or 50% and, at the same time, degrades performance by 5% or 40%. This results in E_TC = 0.95/0.80 or 0.60/0.50 = 1.20 (almost same). Even if the new mapping is more power-performance efficient (with 50%), we will not accept such a severe performance degradation (with 40%).

$$E\_TC = \frac{Perf(mapping\_A)/Perf(mapping\_B)}{Power(mapping\_A)/Power(mapping\_B)} \tag{5.1}$$

Based on this metric, we figured out heuristic thread consolidation approach in Fig. 5.2.

This approach aims to benet from thread consolidation at runtime to minimize power consumption with minimal (or zero) impact on performance. Also, it evaluates E_TC during run time in terms of application's performance and power consumption. In this situation, performance means throughput which represents the total number of instructions completed by all the threads per cycle, and power means chip power consumption. It is triggered every T milliseconds, but makes decisions only if it is enabled. In the initial step, this heuristic approach sets the most unconsolidated mapping for that particular thread bucket to not harm application performance. Before making any consolidation and unconsolidation decision, heuristic thread consolidation method check a bunch of available hardware threads. Based on the number of current software thread (N), a bunch of threads (bucket) is defined as the minimum number of hardware threads (power of 2) required to support the current number of software threads. For example, if the current number of software threads is N = 4, the bucket to be used is 4, or if N = 13, the bucket is 16. By choosing the right bucket, heuristic thread consolidation method knows what are the possible thread mappings it can play with. All possible thread buckets in POWER7 [58] are surveyed: 1, 2, 4, 8, 16 and 32 threads. Once the bucket is chosen, heuristic thread consolidation method sets the most unconsolidated mapping for that particular bucket to not harm application performance. Throughout the example, we will refer to this mapping as mapping_{prev}. The first action of heuristic thread consolidation method is consolidation. Just after a bucket selection, there is no power-performance history because the previous bucket may have a completely different power-performance footprint. Hence, heuristic thread consolidation method begins judging the effects of consolidation on power-performance efciency. We refer to this new mapping as mapping_{next}. To compare mapping_{next} versus mapping_{prev}, E_TC is then computed and analyzed. If E_TC is larger than average profiled E_TC, heuristic thread consolidation method assumes that the application is traversing a consolidation-friendly phase, and further consolidates threads if it is possible. If, instead, E_TC is smaller than average profiled E_TC, heuristic thread consolidation method considers that mapping_{next} is less power-performance

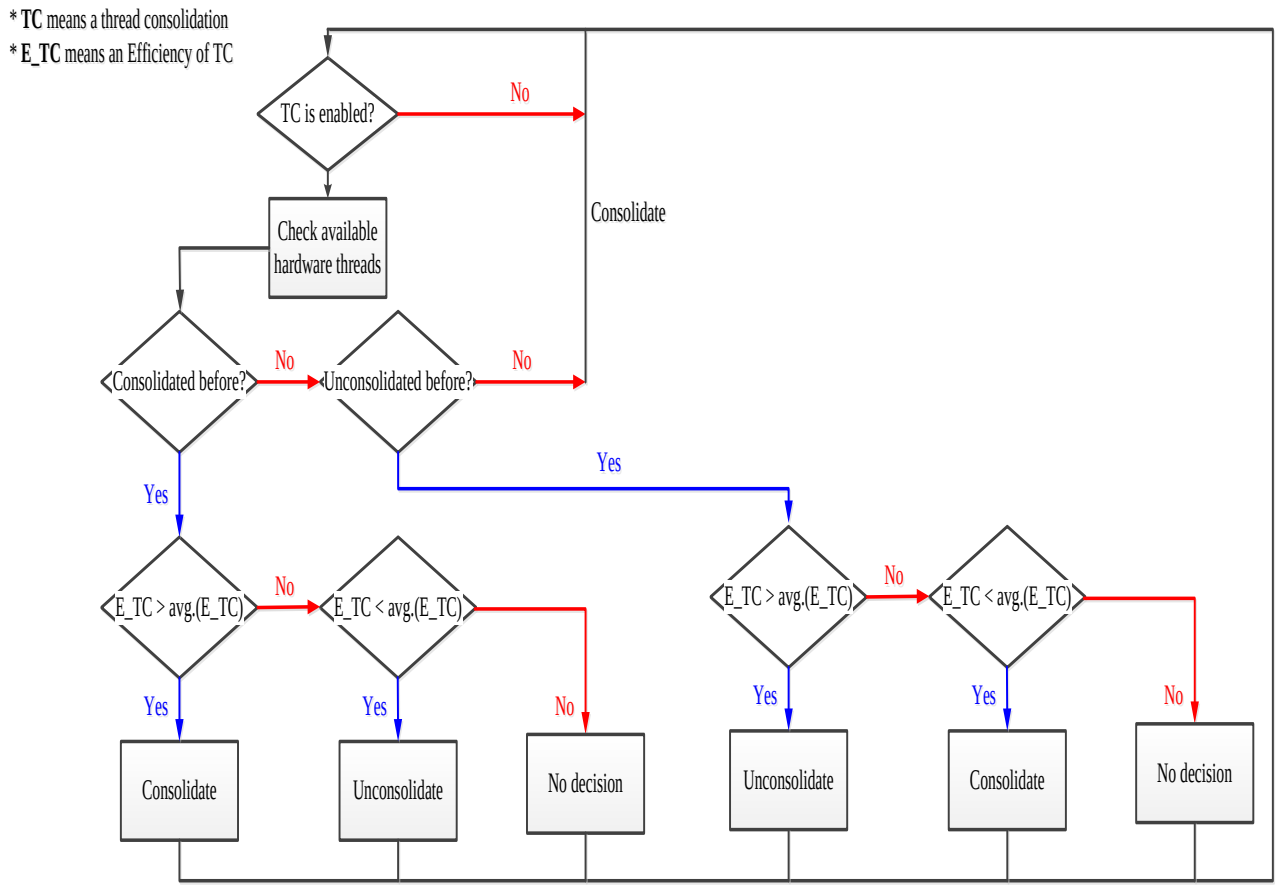


Figure 5.2: Outline of Heuristic Thread Cosolidation Method.

efcient than mapping_{prev}, and goes back to mapping_{prev}. Heuristic thread consolidation method continues in this way, analyzing the E_TC of the last action, and making a new decision based on that. Every time, if E_TC is not changed, heuristic thread consolidation method makes no decision.

Chapter 6

Evaluation And Methodology

In our previous work [15], we simulated the execution of 10 benchmarks from the Rodinia benchmark suite [16]. Benchmarks are simulated using the recent architectural simulator, namely gem5GPU [22], which combined with gem5 [20] and GPGPU-Sim [21] on an unmodified x86 64bits Linux 2.6. They simulated 256 in-order x86 CPU cores at 2GHz and distributed into 16 tiles, and 768 GPU cores at 600MHz and also distributed into 16 tiles. The gem5GPU system obeys traditional MOESI cache coherence protocol [23], and the GPU caches are write-through and obey the VI-based cache coherence protocol. The detail parameters of the simulation infrastructure are reported in Table 6.1.

We get some benefits from this work [15], especially focusing on the design of fusion coherence by two-level directory. It is directly accessing uniformed L3 data cache, which sharply decrease the execution time and latency of accessing memory system. It is due to bypassing L3 data cache guided by fusion directory, directly accessing memory space instead of copying data between each other and transmitting over system bus. As a result, the average speedup is 2.4X and the maximum speedup is more than 4X. However, some benchmarks do not achieve the high speedup. These benchmarks are mainly computing-intensive, and spend a

Table 6.1: Simulation Configuration Parameters

	L1 D-Cache	L1 I-Cache	L2 D-Cache	L3 D-Cache
CPU	64KB	64KB	2MB, 4x4-way	32MB (uniformed), 16-way
GPU	64KB	64KB	8MB, 8x4-way	32MB (uniformed), 16-way
* In-order CPU cores at 2GHz / GPU cores at 600MHz * Number of cores: CPU (768) / GPU (256)				

significant execution time on CPU core instead of transferring a large mount of data between CPU and GPU memory.

Emerging manycore processors have also brought new challenges under available power budgets to the architecture research community. Manycore processors are highly integrated complex system-on-chips with complicated core and uncore subsystems. The core subsystems can consist of a large number of traditional and asymmetric cores. The uncore subsystems have also become unprecedentedly powerful and complex with deeper cache hierarchies, advanced on-chip interconnects, and high-performance memory controllers. In order to conduct research for emerging manycore processor systems, a microarchitecture-level and cycle-level manycore simulation infrastructure is needed. Numerous processor and system simulators are already available. All of these simulators have their own merits and serve their different purposes well. However, according to our specific research goal with regard to power awareness, we try to compare the existing two remarkable simulators such as gem5GPU and Sniper with various dimensions of features as shown in Table 6.2.

Based on Table 6.2, we are able to know that Sniper provides the detail necessary to estimate power across many-core architecture. Sniper [63], [64], [65] is also satisfied with fast and scalable simulation under two major trends in high-performance computing, namely, large numbers of cores and the growing size of on-chip cache memory. By bringing together

Table 6.2: Feature's Summary of Existing Well-Known Simulators

		gem5GPU	Sniper	Which one is more proper?
1	Simulation Speed	Somewhat slow(+)	Faster(+++)	Sniper
2	Simulation Accuracy	N/Av	Within 25%	
3	Power Estimation	N/Ap	Yes	Sniper
4	Many-Core Support	P/S	Yes	Sniper
5	Caches	Y/A	Yes	gem5GPU
6	Heterogeneous Configuration Support	Y/A	Yes	gem5GPU
7	Advanced Visuable Support	N/Ap	Yes	Sniper

* N/Ap: Not Applicable
* N/Av: Not Available
* P/S: Partially Support
* Y/A: Yes with Advanced Future Support

accurate high-abstraction analytical models with fast parallel simulation, architects can trade off accuracy with simulation speed to allow for longer application runs, covering a larger portion of the hardware design space. In addition, Sniper is integrated with McPAT [66], an integrated power, area, and timing modeling framework that supports comprehensive design space exploration for many-core architecture under processor configurations ranging from 90nm to 22nm and beyond. An extended description of these simulators categorized by specific features is introduced, which sections in Appendix A and B on the last part of the thesis.

In our work, we utilize the data from offline profiling of Parsec benchmarks and implement a binning approach to categorize unknown threads as their nearest neighbor in Parsec benchmarks. Furthermore, we plan to extend this profiling by using Rodinia, Spec omp [18], and Splash-2 [19] benchmarks for considering a variety of exceptional cases. Based on some test

cases such as fft, and fft-hetero, we have shown an architectural topology, power consumption, and cycle per instruction (CPI) stack (see Fig. 6.1, 6.2, 6.3) under visualization support to gain insight into lost cycles.

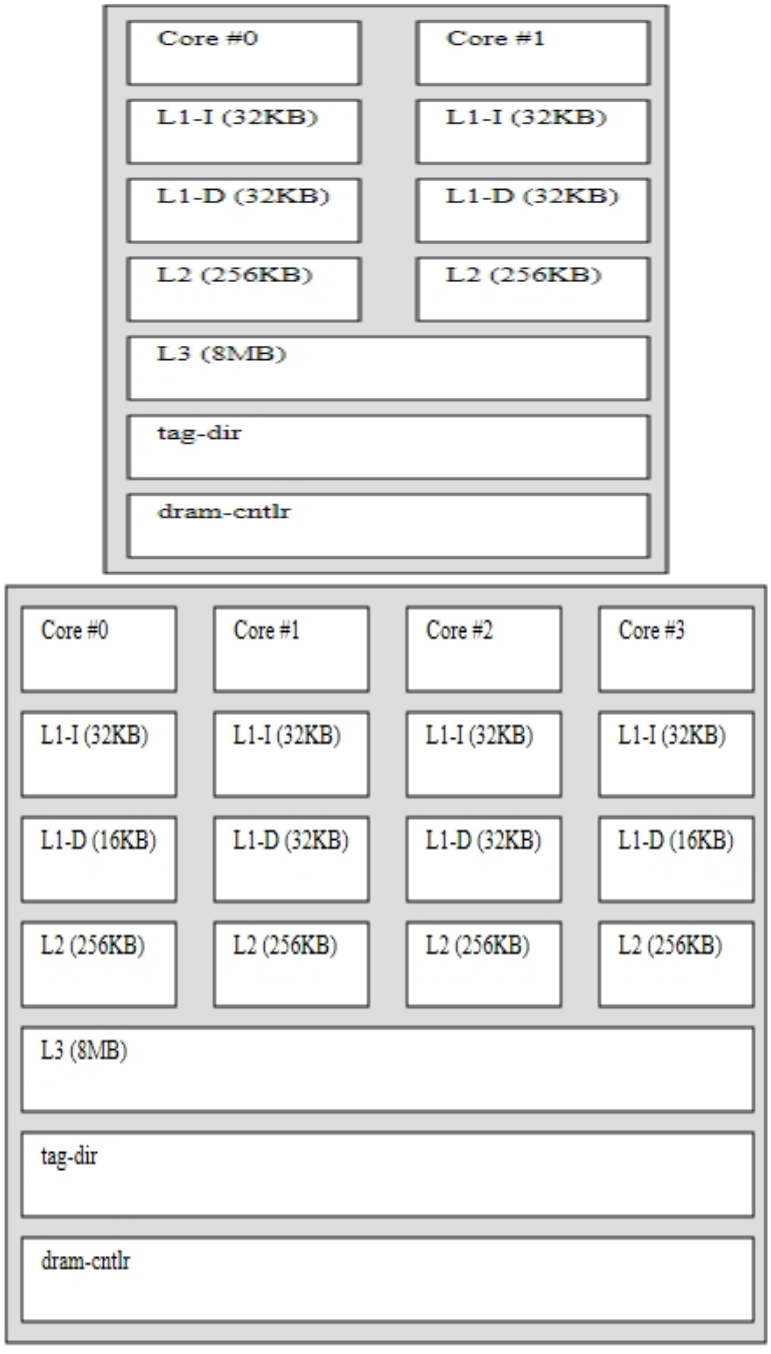


Figure 6.1: Architectural Topology of FFT and FFT-HETERO Test Case. - Generated Results from McPAT framework

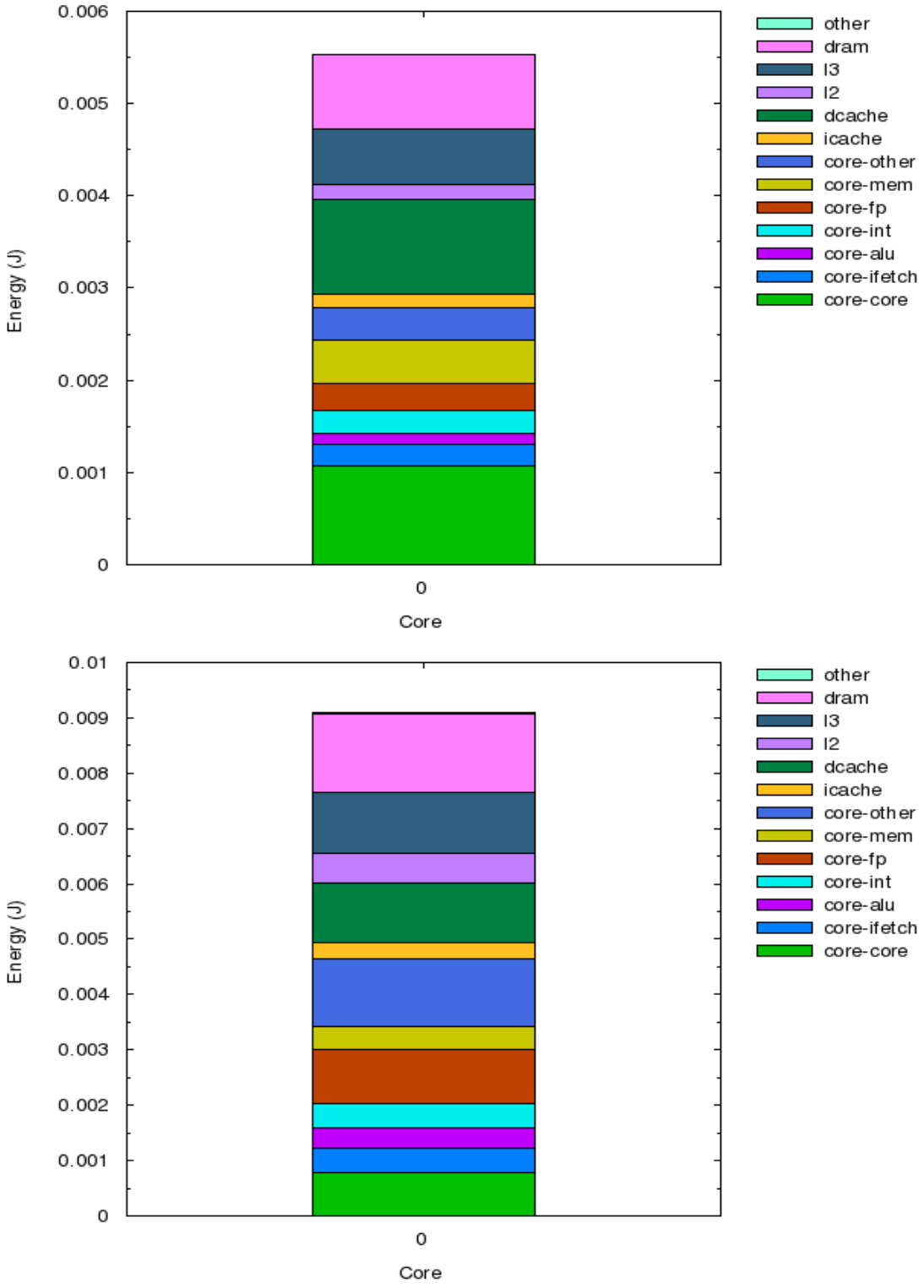


Figure 6.2: Power Consumption of FFT and FFT-HETERO Test Case. - Generated Results from McPAT framework

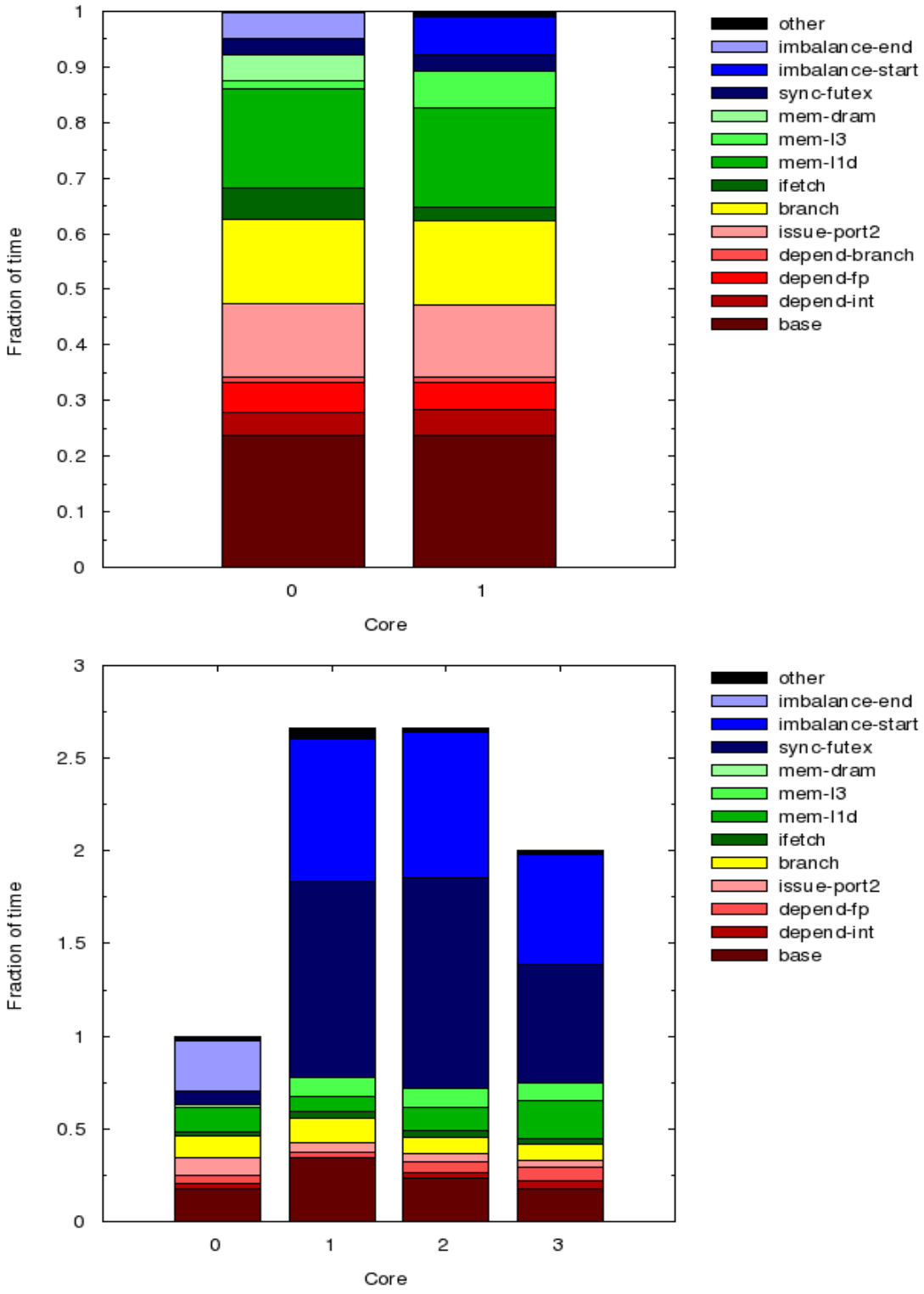


Figure 6.3: CPI Stack of FFT and FFT-HETERO Test Case. - Generated Results from McPAT framework

Chapter 7

Future Work

Power depends not only on the configuration of a processor, but also on the circuit design style and process parameters. Also actual power dissipation varies with activity, though the degree of variability again depends on the technology parameters as well as the gating style used. No existing architecture-level power modeling framework accounts for all of these factors. Therefore, we explore well-known McPAT modeling framework, analyze it, and compare it with Wattch [69], we should at last apply our case study to McPAT in Sniper architectural simulator by a proper modification and estimate our power management scheme in terms of peak and typical power consumption running various applications, and further the chip's area.

Chapter 8

Summary

In this paper, we have introduced and sought to gain some insights into the power benefits available for the future architecture, that of a heterogeneous many-core architecture on the same chip, using a 3-bit core power control scheme and heuristic thread consolidation approach. The particular opportunity examined is for powering a new core up from powered down when we have an application switching among cores. As a result, it reduces a huge latency and a power dissipation for power the core up from powered down. Operation is based on distinct scenarios by 3-bit core power control scheme through 5 statuses switching. In addition, for more elaborated control to be power-performance efficient, this kind of status switching is exactly triggered by power-aware thread placement thorugh heuristic thread consolidation approach.

Bibliography

[1] (2013) Heterogeneous System Architecure (HSA) Foundation. [Online]. Available: http://www.hsafoundation.com

[2] (2014) International Technology Roadmap for Semiconductor (ITRS) Organization. [Online]. Available: http://www.itrs.net

[3] S. Borkar, "Thousand Core Chips: A Technology Perspective", in *Proceedings of Design Automation Conference*, Jun. 2007

[4] N. Brookwood, "AMD Fusion Family of APUs: Enabling a Superior, Immersive PC Experience", AMD White Paper, Jan. 2014.

[5] (2014, Jan.) Intel Haswell Microarchitecture. [Online]. Available: http://www.intel.com

[6] (2014, Jan.) Nvidia Denver Project. [Online]. Available: http://www.nvidia.com

[7] (2014, Jan.) ARM big.LITTLE Processing. [Online]. Available: http://www.arm.com

[8] D. R. Johnson, M. R. Johnson, J. H. Kelm, W. Tuohy, S. S. Lumetta, and S. J. Patel, "Rigel: A 1,024-Core Single-Chip Accelerator Architecture", *IEEE Micro*, Aug. 2011.

[9] R. Kumar, K. I. Farkas, N. P. Jouppi, P. Ranfanathan, and D. M. Tullsen, "A Multi-core Approach to Addressing the Energy-Complexity Problem in Microprocessors", in *Proceedings of Workshop on Complexity-Effective Design*, Jun. 2003.

[10] T. Sherwood, and B. Calder, "Time Varing Behavior of Programs", UC San Diego Technical Report UCSD-CS-99-630, Aug. 1999.

[11] D. Wall,"Limits of Instruction-Level Parallelism", in *Proceedings of International Conference of Architectural Support for Programming Languages and Operating Systems*, Apr. 1991.

[12] T. Pering, T. Burd, and R. Brodersen, "The Simulation and Evaluation of Dynamic Voltage Scaling Algorithm", in *Proceedings of International Symposium on Low Power Electronics and Design*, Aug. 1998.

[13] S. Manne, A. Klauser, and D. Grunwald, "Pipeline Gating: Speculation Control for Energy Reduction", in *Proceedings of International Symposium on Computer Architecture*, Jun. 1998.

[14] M. Ferdman, P. L. Kamran, K. Balet, and B. Falsafi, "Cuckoo Directory: A Scalable Directory for Many-Core Systems", in *Proceedings of International Symposium of High Performance Computer Architecture*, Feb. 2011.

[15] S. Pei, M. S. Kim, J. L. Gaudiot, and N. Xiong, "Fusion Coherence: Scalable Cache Coherence for Heterogeneous Kilo-Core System", in *Proceedings of 10th Annual Conference on Advanced Computer Architecture*, Aug. 2014.

[16] S. Che, J. W. Sheaffer, M. Boyer, L. G. Szafaryn, L. Wang, and K. Skadron, "A Characterization of the Rodinia Benchmark Suite with Comparison to Contemporary CMP Workloads", in *Proceedings of the International Symposium on Workload Characterization*, Dec. 2010.

[17] C. Bienia, "Benchmarking Modern Multiprocessors", Ph.D. Thesis, Princeton University, Jan. 2011.

[18] V. Aslot, M. Domeika, R. Eigenmann, G. Gaertner, W. B. Jones, and B. Parady, "SPEComp: A New Benchmark Suite for Measuring Parallel Computer Performance", in *Proceedings of International Workshop on OpenMP Applications and Tools*, Jul. 2001.

[19] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta, "The SPLASH-2 Programs: Characterization and Methodological Considerations", in *Proceedings of International Symposium on Computer Architecture*, Jun. 1995.

[20] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sowell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, "The Gem5 Simulator", in *Proceedings of International Symposium on Computer Architecture*, Jun. 2011.

[21] A. Bakhoda, G. L. Yuan, W. W. L. Fung, H. Wong, and T. M. Aamodt, "Analyzing CUDA Workloads Using a Detailed GPU Simulator", in *Proceedings of Internation Symposium on Performance Analysis of Systems and Software*, Apr. 2009.

[22] J. Power, J. Hestness, M. S. Orr, M. D. Hill, and D. A. Wood , "gem5-gpu: A Heterogeneous CPU-GPU Simulator", *IEEE Computer Architecture Letters*, Nov. 2013.

[23] D. J. Sorin, M. D. Hill, and D. A. Wood, "A Primer on Memory Consistency and Cache Coherence", Synthesis Lectures on Computer Architecture, Morgan & Claypool Publishers, Nov. 2011.

[24] (2009) HP-Intel Dynamic Power Capping. [Online]. Available: http://techhead.co/hp-dynamic-power-capping/

[25] J. Li, and J. Martinez, "Dynamic Power-Performance Adaptation of Parallel Computation on Chip Multiprocessors", in *International Symposium on High-Performance Computer Architecture*, Feb. 2006.

[26] K. Singh, M. Bhadauria, and S. A. McKee, "Real Time Power Estimation and Thread Scheduling via Performance Counters", *SIGARCH Computer Architecture News*, Jul. 2009.

[27] D. Shin, J. Kim, and S. Lee, "Low-Energy Intra-Task Voltage Scheduling using Static Timing Analysis". in *Proceedings of the Design Automation Conference*, Jun. 2001.

[28] A. Azevedo, I. Issenin, R. Cornea, R. Gupta, N. Dutt, A. Veidenbaum, and A. Nicolau, "Profile-Based Dynamic Voltage Scheduling using Program Checkpoints", in *Proceedings of Design, Automation and Test in Europe Conference*, Mar. 2002.

[29] C. Isci, G. Contreras, and M. Martonosi, "Live, Runtime Phase Monitoring and Prediction on Real Systems with Application to Dynamic Power Management", in *Proceedings of the International Symposium on Microarchitecture*, Dec. 2006.

[30] G. Dhiman and T. S. Rosing, "Dynamic Voltage Frequency Scaling for Multi-Tasking Systems using Online Learning", in *Proceedings of International Symposium on Low Power Electronics and Design*, Aug. 2007.

[31] K. K. Rangan, G.-Y. Wei, and D. Brooks, "Thread Motion: Fine-Grained Power Management for Multi-Core Systems", in *Proceedings of International Symposium on Computer Architecture*, Jun. 2009.

[32] M. Etinski, J. Corbalan, J. Labarta, and M. Valero, "Optimizing Job Performance under a Given Power Constraint in HPC Centers", in *Proceedings of the International Conference on Green Computing*, Aug. 2010.

[33] J. M. Cebrian, J. L. Aragon, and S. Kaxiras, "Power Token Balancing: Adapting CMPs to Power Constraints for Parallel Multithreaded Workloads", in *Proceedings of International Parallel and Distributed Processing Symposium*, May 2011.

[34] C. Isci, A. Buyuktosunoglu, C.-Y. Cher, P. Bose, and M. Martonosi, "An Analysis of Efficient Multi-Core Global Power Management Policies: Maximizing Performance for a Given Power Budget", in *Proceedings of the International Symposium on Microarchitecture*, Dec. 2006.

[35] R. Teodorescu and J. Torrellas, "Variation-Aware Aapplication Scheduling and Power Management for Chip Multiprocessors", in *Proceedings of International Symposium on High-Performance Computer Architecture*, Jun. 2008.

[36] G. Magklis, M. L. Scott, G. Semeraro, D. H. Albonesi, and S. Dropsho, "Profile-Based Dynamic Voltage and Frequency Scaling for a Multiple Clock Domain Microprocessor", in *Proceedings of International Symposium on Computer Architecture*, Jun. 2003.

[37] S. Herbert and D. Marculescu, "Analysis of Dynamic Voltage/Frequency Scaling in Chip-Multiprocessors", in *Proceedings of International Symposium on Low Power Electronics and Design*, Aug. 2007.

[38] W. Kim, M. Gupta, G.-Y. Wei, and D. Brooks, "System Level Analysis of Fast, Per-Core DVFS using On-Chip Switching Regulators", in *Proceedings of International Symposium on High Performance Computer Architecture*, Feb. 2008.

[39] J. M. Cebrian, J. L. Aragon, and S. Kaxiras, "Power Token Balancing: Adapting CMPs to Power Constraints for Parallel Multithreaded Workloads", in *Proceedings of International Parallel and Distributed Processing Symposium*, May 2011.

[40] J. Sartori and R. Kumar, "Distributed Peak Power Management for Many-Core Architectures", in in *Proceedings of Design, Automation and Test in Europe Conference*, Apr. 2009.

[41] A. Gandhi, R. Das, J. O. Kephart, M. Harchol-balter, and C. Lefurgy, "Power Capping via Forced Idleness", in *Proceedings of Workshop on Energy Efficient Design*, Jun. 2009.

[42] K. Meng, R. Joseph, and R. P. Dick, "Multi-Optimization Power Management for Chip Multiprocessors", in *Proceedings of International Conference on Parallel Architectures and Compilation Techniques*, Oct. 2008.

[43] V. Kontorinis, A. Shayan, D. M. Tullsen, and R. Kumar, "Reducing Peak Power with a Table-Driven Adaptive Processor Core", in *Proceedings of the International Symposium on Microarchitecture*, Dec. 2009.

[44] (2009) AMD Brings Power Capping to New 45nm Opteron Line. [Online]. Available: http://www.infoworld.com/d/green-it/amd-brings-power-capping-new-45nm-opteron-line-906

[45] D. Meisner, B. T. Gold, and T. F. Wenisch, "PowerNap: Eliminating Server Idle Power", in *Proceeding of the International Conference on Architectural Support for Programming Languages and Operating Systems*, Mar. 2009.

[46] H. Li, C. Cher, T. N. Vijaykumar, and K. Roy, "Combined Circuit and Architectural Level Variable Supply-Voltage Scaling for Low Power", *IEEE Transactions on Very Large Scale Integration Systems*, May 2005.

[47] P. Juang, Q. Wu, L. Peh, M. Martonosi, and D. Clark, "Coordinated, Distributed, Formal Energy Management of Chip Multiprocessors", in *Proceedings of International Symposium on Low Power Electronics and Design*, Aug. 2005.

[48] S. Heo, K. Barr, and K. Asanovic, "Reducing Power Density through Activity Migration", in *Proceedings of International Symposium on Low Power Electronics and Design*, Aug. 2003.

[49] M. D. Powell, M. Gomaa, and T. Vijaykumar, "Heat-and-Run: Leveraging SMT and CMP to Manage Power Density Through the Operating System", in *Proceedings of International Conference on Architectural Support for Programming Languages and Operating Systems*, Oct. 2004.

[50] A. Shayesteh, E. Kursun, T. Sherwood, S. Siar, and G. Reinman, "Reducing the Latency and Area Cost of Core Swapping through Shared Helper Engines", in *Proceedings of IEEE International Conference on Computer Design*, Oct. 2005.

[51] B. Calhoun and A. Chandrakasan, "Ultra-Dynamic Voltage Scaling (UDVS) using Sub-Threshold Operation and Local Voltage Dithering", *IEEE Journal of Solid-State Circuits*, Jan. 2006.

[52] R. Kalla, B. Sinharoy, W. J. Starke, and M. Floyd, "POWER7: IBMs Next Generation Server Processor", *IEEE Micro*, Mar. 2010.

[53] A. Vega, A. Buyuktosunoglu, and P. Bose, "SMT-Centric Power-Aware Thread Placement in Chip Multiprocessor", in *Proceedings of International Conference on Parallel Architectures and Compilation Techniques*, Sep. 2013.

[54] A. Alameldeen, and D. Wood, "IPC Considered Harmful for Multiprocessor Workloads", *IEEE Micro*, Jul. 2006.

[55] J. Leverich, M. Monchiero, V. Talwar, P. Ranganathan, and C. Kozyrakis, "Power Management of Datacenter Workloads using Per-Core Power Gating", *IEEE Computer Architecture Letters*, Feb. 2009.

[56] D. Burgess, E. Gieske, J. Holt, T. Hoy, and G. Whisenhunt, "e6500: Freescales Low-Power, High-Performance Multithreaded Embedded Processor", *IEEE Micro*, Sep. 2012.

[57] (2008) Intel Nehalem Microarchitecture. [Online]. Available: http://www.intel.com/content/dam/doc/white-paper/intel-microarchitecture-white-paper.pdf

[58] S. Taylor, "POWER7+^{TM}: IBMs Next Generation POWER Microprocessor", in *Proceedings of Hot Chips Symposium*, Aug. 2012.

[59] R. Cochran, C. Hankendi, A. K. Coskun, and S. Reda, "Pack & Cap: Adaptive DVFS and Thread Packing under Power Caps", in *Proceedings of the International Symposium on Microarchitecture*, Dec. 2011.

[60] D. Tam, R. Azimi, and M. Stumm, "Thread Clustering: Sharing-Aware Scheduling on SMP-CMP-SMT Multiprocessors", in *Proceedings of European Conference on Computer Systems*, Jun. 2007.

[61] K. K. Rangan, G.-Y. Wei, and D. Brooks, "Thread Motion: Fine-Grained Power Management for Multi-Core Systems", in *Proceedings of International Symposium on Computer Architecture*, Jun. 2009.

[62] A. Bhattacharjee and M. Martonosi, "Thread Criticality Predictors for Dynamic Performance, Power, and Resource Management in Chip Multiprocessors", in *Proceedings of International Symposium on Computer Architecture*, Jun. 2009.

[63] T. E. Carlson, W. Heirman, and L. Eeckhout, "Sniper: Exploring the Level of Abstraction for Scalable and Accurate Parallel Multi-Core Simulation", in *Proceedings of International Conference for High Performance Computing, Networking, Storage and Analysis*, Nov. 2011.

[64] T. E. Carlson, W. Heirman, and L. Eeckhout, "Sampled Simulation of Multi-Threaded Applications", in *Proceedings of International Symposium on Performance Analysis of Systems and Software*, Apr. 2013.

[65] T. E. Carlson, W. Heirman, S. Eyerman, I. Hur, and L. Eeckhout, "An Evaluation of High-Level Mechanistic Core Models", *ACM Transactions on Architecture and Code Optimization*. Jul. 2014.

[66] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "McPAT: An Integrated Power, Area, and Timing Modeling Framework for Multicore

and Manycore Architectures", in *Proceedings of the International Symposium on Microarchitecture*, Dec. 2009.

[67] D. Genbrugge, S. Eyerman, and L. Eeckhout, "Interval Simulation: Raising the Level of Abstraction in Architectural Simulation", in *Proceedings of International Symposium on High Performance Computer Architecture*, Jan. 2010.

[68] J. E. Miller, H. Kasture, G. Kurian, C. Gruenwald III, N. Beckmann, C. Celio, J. Eastep and A. Agarwal, "Graphite: A Distributed Parallel Simulator for Multicores", in *Proceedings of International Symposium on High Performance Computer Architecture*. Jan. 2010.

[69] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: A Framework for Architectural-Level Power Analysis and Optimizations", in *Proceedings of International Symposium on Computer Architecture*, Jun. 2000.

[70] S. Thoziyoor, J. H. Ahn, M. Monchiero, J. Brockman, and N. Jouppi, "A Comprehensive Memory Modeling Tool and its Application to the Design and Analysis of Future Memory Hierarchies", in *Proceedings of International Symposium on Computer Architecture*, Jun. 2008.

[71] A. Kahng, B. Li, L. S. Peh, and K. Samadi, "ORION 2.0: A Fast and Accurate NoC Power and Area Model for Early-Stage Design Space Exploration", in *Proceedings of Design, Automation and Test in Europe Conference*, Apr. 2009.

Appendix A

Sniper: Scalable and Accurate Parallel Multi-Core Simulator

Sniper is a next generation parallel, high-speed and accurate x86 simulator. This multi-core simulator is based on the interval core model [67] and the Graphite simulation infrastructure [68], allowing for fast and accurate simulation and for trading off simulation speed for accuracy to allow a range of flexible simulation options when exploring different homogeneous and heterogeneous multi-core architectures.

The Sniper simulator allows one to perform timing simulations for both multi-program workloads and multi-threaded, shared-memory applications with 10s to 100+ cores, at a high speed when compared to existing simulators. The main feature of the simulator is its core model which is based on interval simulation, a fast mechanistic core model. Interval simulation raises the level of abstraction in architectural simulation which allows for faster simulator development and evaluation times; it does so by 'jumping' between miss events, called intervals. Sniper has been validated against multi-socket Intel Core2 and Nehalem systems and provides average performance prediction errors within 25% at a simulation speed of up to

several MIPS.

This simulator, and the interval core model, is useful for uncore and system-level studies that require more detail than the typical one-IPC models, but for which cycle-accurate simulators are too slow to allow workloads of meaningful sizes to be simulated. As an added benefit, the interval core model allows the generation of CPI stacks, which show the number of cycles lost due to different characteristics of the system, like the cache hierarchy or branch predictor, and leads to a better understanding of each component's effect on total system performance. This extends the use for Sniper to application characterization and hardware/software co-design.

A.1 Intel Nehalem Architecture

Nehalem-based microprocessors, as can be seen in Figure A.1 and Figure A.2, use the 45nm process, run at higher clock speeds, and are more energy-efficient than the older microprocessor. Hyper-threading is reintroduced, along with a reduction in L2 cache size, as well as an enlarged L3 cache that is shared among all cores. It involved some technologies as shown in Table A.1.

It has been reported that Nehalem has a focus on performance, thus the increased core size. Compared to the older microprocessor, Nehalem has 10-25% better single-threaded performance / 20-100% better multithreaded performance at the same power level 30% lower power consumption for the same performance. On average, Nehalem provides a 15-20% clock-for-clock increase in performance per core. Overclocking is possible. Nehalem processors incorporate SSE 4.2 SIMD instructions, adding seven new instructions to the SSE 4.1 set in the Core 2 series. The Nehalem architecture reduces atomic operation latency by 50% in an attempt to eliminate overhead on atomic operations.

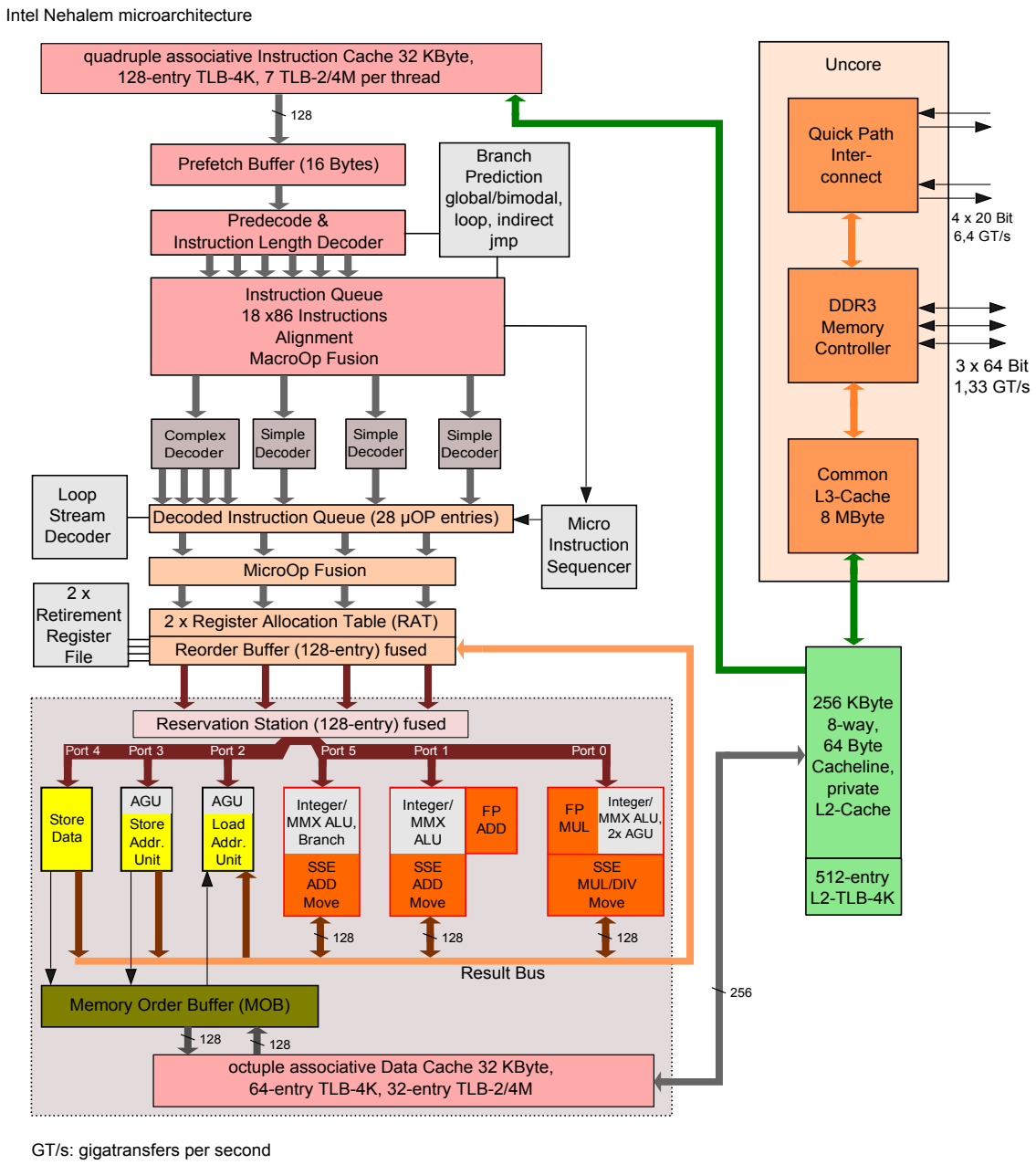


Figure A.1: Nehalem Microarchitecture. - Image from Nehalem Processor at Intel.com

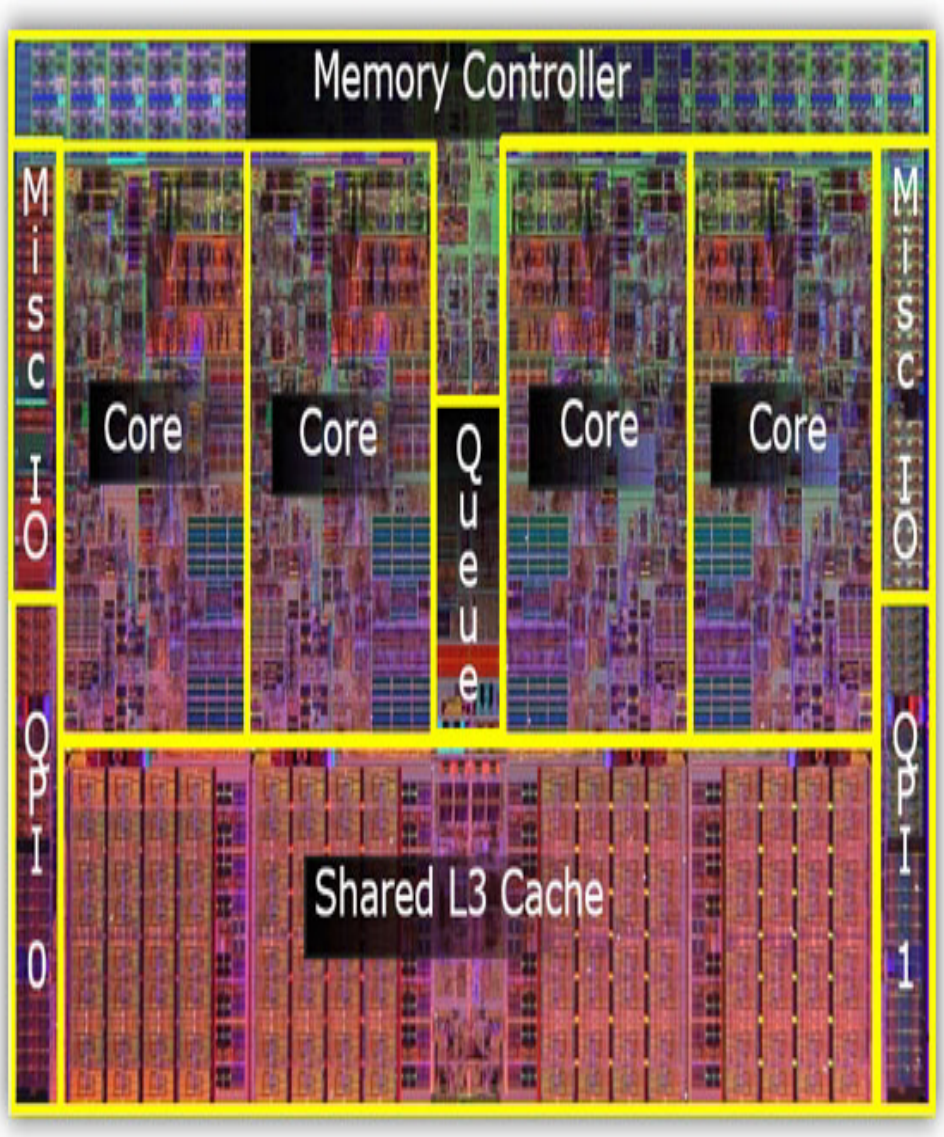


Figure A.2: Nehalem Core Die. - Image from Nehalem Processor at Intel.com

Table A.1: Technology of Nehalem

Technology		
Hyper-Threading	Reintroduced	
Cache	L1	64KB Per Core
	L2	256KB Per Core
	L3	4MB to 24MB Shared
Second Level	Branch Predictor Translation Lookaside Buffer (TLB)	
Interconnect	Quickpath	
ETC	Integration of PCI Express and DMI Integrated Memory Controller 20 to 24 Pipeline Stages	

TLB Sizes			
Cache		Page Sizes	
Name	Level	4KB	2MB
DTLB	1st	64	32
ITLB	1st	128	7 / Logical core
STLB	2nd	512	None

A.2 Interval Simulation

Interval simulation is a recently proposed simulation approach for simulating multi-core and multiprocessor systems at a higher level of abstraction compared to current practice of detailed cycle-accurate simulation. This technique leverages a mechanistic analytical model to abstract core performance by driving the timing simulation of an individual core without the detailed tracking of individual instructions through the core's pipeline stages. As shown in Fig. A.3, the foundation of the model is that miss events (like branch mispredictions, cache and TLB misses, serialization instructions, etc.) divide the smooth streaming of instructions through the pipeline into intervals. Branch predictor, memory hierarchy, cache coherence and interconnection network simulators determine the miss events; the analytical model derives

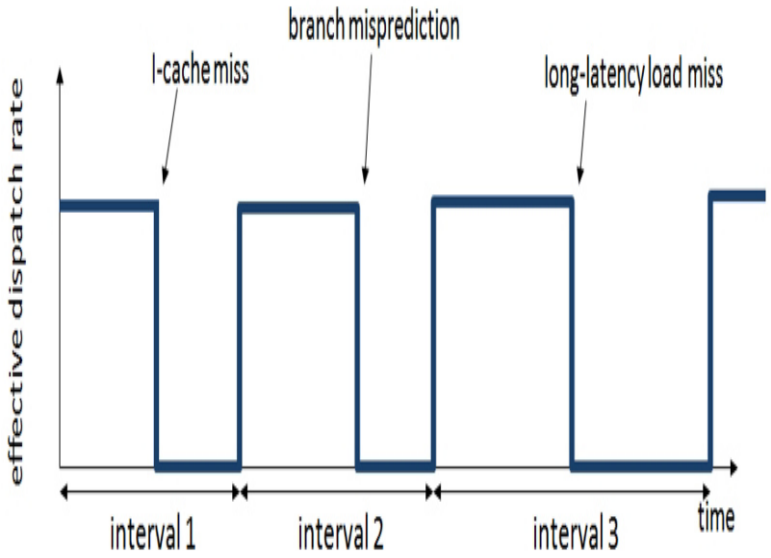


Figure A.3: Interval Simulation. - Image from Sniper Simulator at Snipersim.org

the timing for each interval. The cooperation between the mechanistic analytical model and the miss event simulators enables the modeling of the tight performance entanglement between co-executing threads on multi-core processors.

A.3 Multi-Core Interval Simulator

Fig. A.4 shows the multi-core interval simulator which models the timing for the individual cores. The simulator maintains a window of instructions for each simulated core. This window of instructions corresponds to the reorder buffer of a superscalar out-of-order processor, and is used to determine miss events that are overlapped by long-latency load misses. The functional simulator feeds instructions into this window at the window tail. Core-level progress (i.e., timing simulation) is derived by considering the instruction at the window head. In case of an I-cache miss, the core simulated time is increased by the miss latency. In case of a branch misprediction, the branch resolution time plus the front-end pipeline depth is added to the core simulated time, i.e., this is to model the penalty for executing the

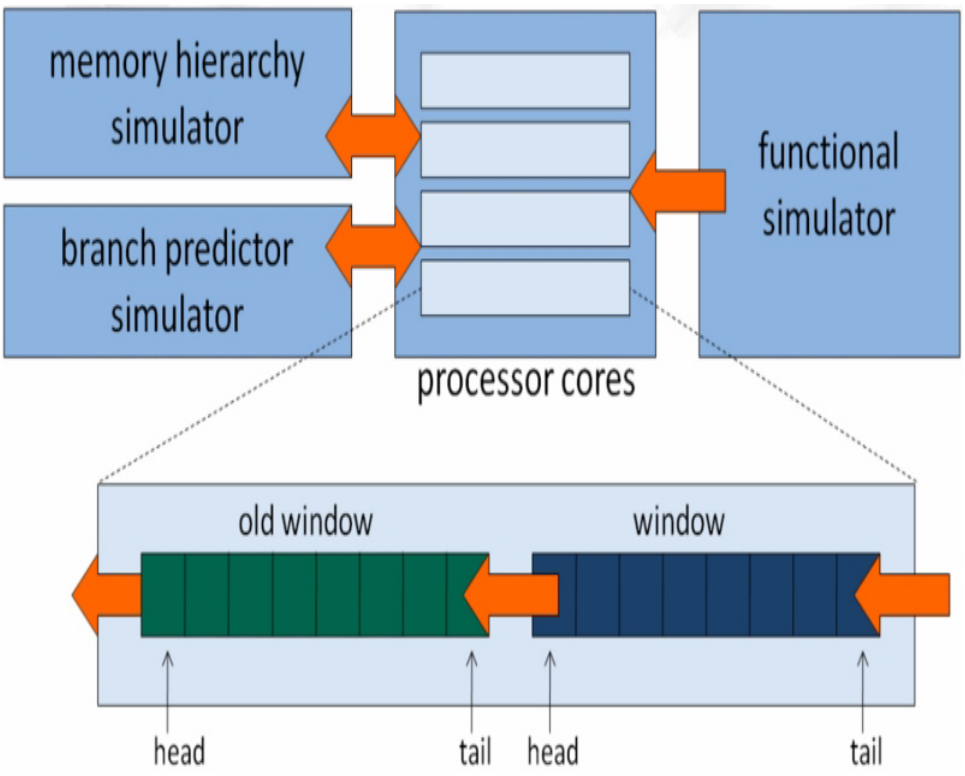


Figure A.4: Multi-Core Interval Simulator. - Image from Sniper Simulator at Snipersim.org

chain of dependent instructions leading to the mispredicted branch plus the number of cycles needed to refill the front-end pipeline. In case of a long-latency load (i.e., a last-level cache miss or cache coherence miss), we add the miss latency to the core simulated time, and we scan the window for independent miss events (cache misses and branch mispredictions) that are overlapped by the long-latency load-second-order effects. For a serializing instruction, we add the window drain time to the simulated core time. If none of the above cases applies, we dispatch instructions at the effective dispatch rate, which takes into account inter-instruction dependencies as well as their execution latencies.

A.4 Instruction-Window Centric Core Model

Large core counts and complex cache hierarchies are increasing the burden placed on commonly used simulation and modeling techniques. Although analytical models provide fast results, they do not apply to complex, many-core shared-memory systems. In contrast, detailed cycle-level simulation can be accurate but also tends to be slow, which limits the number of configurations that can be evaluated. A middle ground is needed that provides for fast simulation of complex many-core processors while still providing accurate results.

In the group of snipersim.org, they explore, analyze, and compare the accuracy and simulation speed of high-abstraction core models as a potential solution to slow cycle-level simulation. We describe a number of enhancements to interval simulation to improve its accuracy while maintaining simulation speed. In addition, we introduce the instruction-window centric (IW-centric) core model, a new mechanistic core model that bridges the gap between interval simulation and cycle-accurate simulation by enabling high-speed simulations with higher levels of detail. We also show that using accurate core models like these are important for memory subsystem studies, and that simple, naive models, like a one-IPC core model, can lead to misleading and incorrect results and conclusions in practical design studies. Validation against real hardware shows good accuracy, with an average single-core error of 11.1% and a maximum of 18.8% for the IW-centric model with a 1.5X slowdown compared to interval simulation.

Appendix B

McPAT: Power Analysis Framework for Multi-Core Architectures

Sniper integrates the Multicore Power, Area, and Timing (McPAT) framework for power and area specific modeling for of manycore architectures. The McPAT that is installed with Sniper is based on McPAT which is available from Hewlett-Packard Labs, for increased performance and functionality. McPAT is a new power, area, and timing modeling framework that enables architects to estimate new metrics combining performance with both power and area, which are useful to quantify the cost of new architectural ideas. At the microarchitectural level, McPAT includes models for the fundamental components of a chip multiprocessor, including in-order and out-of-order processor cores, networks-on-chip, shared caches, integrated memory controllers, and multiple-domain clocking. At the circuit and technology levels, McPAT supports critical-path timing modeling, area modeling, and dynamic, short-circuit, and leakage power modeling for each of the device types forecast in the ITRS roadmap including bulk CMOS, SOI, and double-gate transistors. McPAT has a exible XML interface to facilitate its use with many performance simulators. Combined with a performance simulator, McPAT enables architects to consistently quantify the cost of new ideas

and assess tradeoffs of dierent architectures using new metrics like energy-delay-area^2 product (EDA^{2}P) and energy-delay- area product (EDAP). This paper explores the interconnect options of future manycore processors by varying the degree of clustering over generations of process technologies. Clustering will bring interesting tradeoffs between area and performance because the interconnects needed to group cores into clusters incur area overhead, but many applications can make good use of them due to synergies of cache sharing. Combining power, area, and timing results of McPAT with performance simulation of Parsec benchmarks at the 22nm technology node for both common in-order and out-of-order manycore designs shows that when die cost is not taken into account clustering 8 cores together gives the best energy-delay product, whereas when cost is taken into account configuring clusters with 4 cores gives the best EDA^{2}P and EDAP.

B.1 Operation

Rather than being hardwired to a particular simulator, McPAT uses an XML-based interface with the performance simulator. This interface allows both the specication of the static microarchitecture configuration parameters and the passing of dynamic activity statistics generated by the performance simulator. McPAT can also send runtime power dissipation back to the performance simulator through the XML-based interface, so that the performance simulator can react to power (or even temperature) data. This approach makes McPAT very exible and easily ported to other performance simulators. McPAT runs separately from a simulator and only reads performance statistics from it. Performance simulator overhead is minor - only the possible addition of some performance counters. Since McPAT provides complete hierarchical models from the architecture to the technology level, the XML interface also contains circuit implementation style and technology parameters that are specic to a particular target processor. Examples are array types, crossbar types, and the CMOS

technology generation with associated voltage and device types.

B.2 Related Work

Cacti [70] was the first tool to address the need for rapid power, area, and timing estimates for computer architecture research, focusing on RAM-based structures. Cacti uses the method of logical effort to size transistors. It contains optimization features that enable the tool to find a configuration with minimal power consumption, given constraints on area and timing. Using generic circuit models for pipeline stages, it estimates the RC delay for each stage and determines the critical path.

Wattch is a widely-used processor power estimation tool. Wattch calculates dynamic power dissipation from switching events obtained from an architectural simulation and capacitance models of components of the microarchitecture. Wattch has enabled the computer architecture research community to explore power-ecient design options, as technology has progressed; however, limitations of Wattch have become apparent. First, Wattch models power without considering timing and area. Second, Wattch only models dynamic power consumption; the Hot Leakage package partially addressed this deciency by adding models for subthreshold leakage. Third, Wattch uses simple linear scaling models based on 80nm technology that are inaccurate to make predictions for current and future deep-submicron technology nodes.

Orion [71] is a tool for modeling power in networks-on-chip (NoC). Orion includes models for area, dynamic power, and gate leakage, but does not consider short-circuit power or timing.