November 2016

Dynamic Processor Reconfiguration for Power, Performance and Reliability Management

Sudarshan Srinivasan
University of Massachusetts Amherst

DYNAMIC PROCESSOR RECONFIGURATION FOR POWER, PERFORMANCE AND RELIABILITY MANAGEMENT

A Dissertation Presented

by

SUDARSHAN SRINIVASAN

Submitted to the Graduate School of the
University of Massachusetts Amherst in partial fulfillment
of the requirements for the degree of

DOCTOR OF PHILOSOPHY

September 2016

Electrical and Computer Engineering

DYNAMIC PROCESSOR RECONFIGURATION FOR POWER, PERFORMANCE AND RELIABILITY MANAGEMENT

A Dissertation Presented

by

SUDARSHAN SRINIVASAN

Approved as to style and content by:

Sandip Kundu, Co-chair

Israel Koren, Co-chair

David Irwin, Member

Prashant Shenoy, Member

Christopher V. Hollot, Department Chair
Electrical and Computer Engineering

ACKNOWLEDGMENTS

First and foremost, I would like to thank Prof. Sandip Kundu for his encouragement, support and constant guidance throughout my graduate studies in the last 7 years. None of the work in this thesis would have been possible without him. Prof. Kundu's has shared his vast knowledge in the different areas of processor design all through my grad life, which has contributed towards this thesis. He has been very approachable all through my graduate years for research discussions and has made sure I stay on the right path in my research. I thank him for helping me graduate as a better researcher and his non-technical discussion in various other aspects has helped me develop my career and as a person in grad school.

I would also thank Prof.Israel Koren who has also contributed significantly in shaping up this thesis. I thank him for all the critical reasoning about my work, which has led to better reasoning of my research. His attention to details in all aspects of research has contributed significantly in submitting better quality publications and I would like to thank him for that.

I would also like to thank Professors David Irwin and Prashant Shenoy for their service on my thesis committee. I also extend my gratitude to Rance, Arun, Arunachalam, Kunal and other past members of the lab for participating in several discussions regarding my research in the last several years. I also thank my friends and family for their constant support and encouragement.

ABSTRACT

DYNAMIC PROCESSOR RECONFIGURATION FOR POWER, PERFORMANCE AND RELIABILITY MANAGEMENT

SEPTEMBER 2016

SUDARSHAN SRINIVASAN

B.E.E., VIT UNIVERSITY

M.Sc., UNIVERSITY OF MASSACHUSETTS AMHERST

Ph.D., UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Professor Sandip Kundu and Professor Israel Koren

Technology advancements allowed more transistors to be packed in a smaller area, while the improved performance helped in achieving higher clock frequencies. This, unfortunately led to a power density problem, forcing processor industry to lower the clock frequency and integrate multiple cores on the same die. Depending on core characteristics, the multiple cores in the die could be symmetric or asymmetric. Asymmetric multi-core processors (AMPs) have been proposed as an alternative to symmetric multi-cores to improve power efficiency. AMPs comprise of cores that implement the same ISA, but differ in performance and power characteristics due to varying sizes of micro-architectural resources. As the computational bottleneck of a workload shifts from one resource to another during its course of execution, reassigning it to another core (where it runs more efficiently), can improve the overall power efficiency. Thus achieving high power efficiency in AMPs requires (i) a diverse

set of cores that are optimized for various program phases, (ii) runtime analysis to determine the best core to run on, and (iii) low overhead of re-assigning a thread to a different core type.

Decisions to swap threads between AMPs are made at coarse grain granularity of millions of instructions, to mitigate the impact of thread migration overhead. But the computational needs of the program rapidly change during the course of its execution. The best core configuration for an application such that, both power consumption and performance are optimized, changes over time rapidly at fine granularity of thousands of instructions. This dissertation explores ways to design core micro-architecture such that high power efficiency could be achieved, if switching overhead could be lowered, enabling fine grain switching.

To take advantage of power saving opportunities at fine grain granularity, this thesis explores reconfigurable/morphable architectures where core resources are reconfigured on demand to suit the needs of the executing application. At first, we explore reconfigurable architectures consisting of two kinds of cores: out-of-order (OOO) big cores and in-order (InO) small cores. The big cores provide higher performance while the small cores are more power efficient. In this proposed architecture, OOO core reconfigures into InO core at run time. Our proposed online management scheme decides to switch between these core types such that we obtain significant power benefits without impacting performance. We also observe that, resource requirements of applications can be quite diverse and consequently, resource bottlenecks or excesses can vary considerably. Thus, reconfiguration between just two core modes may not fully exploit power and performance improvement opportunities.

We therefore, explore reconfigurable architectures consisting of diverse core types that not limited to big and little cores. A single core can reconfigure into multiple core modes where each mode has unique power and performance characteristics. Workload performance on a particular core mode depends on a large set of processor resources.

Some workloads are highly memory intensive, some exhibit large instruction dependency, some experience high rates of branch mis-prediction, while other workloads exhibit large exploitable instruction level parallelism. A diverse set of core modes is needed, that could address shifting resource needs during various program phases of an application. Different trade-offs in power and performance could be achieved by reducing or expanding the size of various resource. Trade-offs for each core mode are also affected by operating voltage and frequency. We therefore, propose joint core resource resizing with dynamic voltage and frequency scaling (DVFS), which is important for applications whose performance is sensitive to changes in frequency. Thus, at fine granularity, the core should adapt to varying instruction window sizes, execution bandwidth and frequency to meet the demands of the workload at run-time to improve power efficiency.

Many current processors employ DVFS aggressively to improve power efficiency and maximize performance. This dissertation studies the tradeoff in power efficiency in using fine grain DVFS and reconfigurable architectures mentioned above.

We also explore another important problem due to continued scaling of devices which results in higher vulnerability to soft-errors. We consider dynamic core reconfiguration from the perspectives of both power efficiency and vulnerability to soft-errors. An online management scheme is proposed such that core reconfiguration upon a thread switch not only improves power efficiency but also does not increase the vulnerability to soft errors.

In summary, we propose in this thesis several solutions for improving power efficiency by integrating heterogeneity within the core. We also address how popular power reduction techniques like DVFS are comparable to our approach. Finally, we address reliability challenges along with improving power efficiency.

TABLE OF CONTENTS

LIST OF TABLES

Table **Page**

LIST OF FIGURES

Figure **Page**

CHAPTER 1

INTRODUCTION

Advancements in technology has resulted in increased transistor performance and the ability to pack more transistors into a small area. We now commonly see processors with beyond 1B transistors on a die as shown in Figure 1.1. Unfortunately the power consumption per transistor hasn't dropped at a corresponding rate [28]. This is primarily due to fundamental physical limits reached at the transistor level. As transistor becomes smaller, leakage current increases because transistor threshold voltages have been reduced to the point where the devices don't completely shut off. The increased device density and rising frequency unfortunately led to a rapid growth in power density. This growth in power density has become unsustainable at 100 W/cm^2 due to packaging limitations. Thus, the industry can no longer rely solely on manufacturing improvements to achieve better power efficiency, forcing them to look at micro-architecture innovations and power management techniques to keep within the power dissipation limits [14].

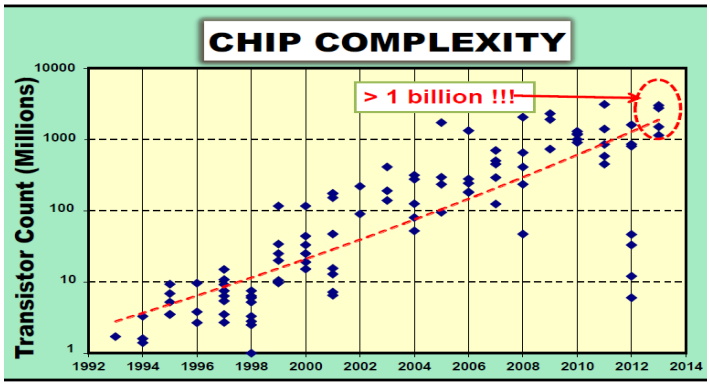

Figure 1.1. Trend in transistor integration on single chip [82]

.

A parallel trend has been to lower the processor frequency and include more cores on the die, which paved the way for the multicore era [39]. Traditionally multicores are symmetric, where all cores have identical power/performance characteristics. Though multicores have resulted in lower power density, they are still constrained by the total power dissipation that depends on the currently available packaging and cooling technologies. Due to limitation on power budget, the fraction of cores in a given multicore design that can run at full speed simultaneously is dropping exponentially [97]. This lead industry/researchers to coin the term 'Dark silicon', where at any given time instance, the number of cores that can be active is limited by chip's power budget and is smaller than the total number of cores that are present in the chip [28, 34]. Dark silicon means that a significant fraction of cores need to be idle (Dark) or under-clocked (Dim) at any point of time. To maintain the power budget, logic is severely under clocked in case of dim silicon. Dim silicon technology includes spatial dimming through use of Dynamic Voltage and Frequency Scaling (DVFS), temporal dimming through computational sprinting [76] or Intel Turbo Boost technology [20] where the chip power budget is allowed to exceed its limit for short durations of time to achieve enhanced performance followed by a long period of inactivity or low power computation. In the dark slicon era, as silicon area is cheap, architects can spend area to increase the energy efficiency. This lead to emergence of Heterogeneous architectures or Asymmetric Multi-core Processors (AMPs) consisting of diverse core types on the die as a potential solution to the power density problem.

AMP architectures are classified into 1) physical asymmetric cores that execute the same ISA but have different micro architectures resulting in diverse power and performance characteristics to better match various application behaviors [9, 35, 40, 52, 54] 2) hybrid cores consisting of different architecture and ISA 3) DVFS to emulate AMP with physical symmetric cores [3] and 4) Reconfigurable cores where a single

core reconfigures into multiple core types with varied frequency and micro-architecture [61, 90, 91]. This thesis is focusing on reconfigurable AMP cores, DVFS to emulate AMP and physical asymmetric cores as explained later in this chapter. As applications exhibit diverse program behavior during the course of execution, AMP architectures offer opportunities to achieve higher power efficiency by dynamically migrating an application from one core type to another based on the current resource needs of the application. Usage of AMP have also gained importance in mobile systems and data centers. In mobile systems, low power consumption helps in prolonging battery life. Data centers experience low average utilization but also periods of high activity that need more computing resources to meet the desired quality of service.

In this thesis we consider AMP architectures consisting of monotonic or non-monotonic core types. Monotonic core types in AMP include high performance/high power and low performance/low power core types [55, 95]. Figure 1.2 shows an example of industry standard monotonic core type, ARM's Big.LITTLE [35]. Big cores (higher performance cores) are used for compute intensive tasks and little cores (low performance cores) are used for less demanding tasks. Another example of monotonic core type is Nvidia's Kal-El [70] which contains 4 high performance and one low power core.

Another class of AMP architectures consists of non-monotonic core types where each core is power/performance optimized to different instruction level behavior of the workload. Non-monotonic core types, thus can provide higher power efficiency than monotonic core types [66, 67, 69]. In Figure 1.3 we show the different cores types and configurations which are analyzed in this thesis and we study the performance and power/trade off in each of the different core configuration.

To take advantage of the performance/power benefits of non-monotonic core types, accurate steering of an application phase to the most suitable core type is required. High performance per watt for single threaded applications requires that we maxi-

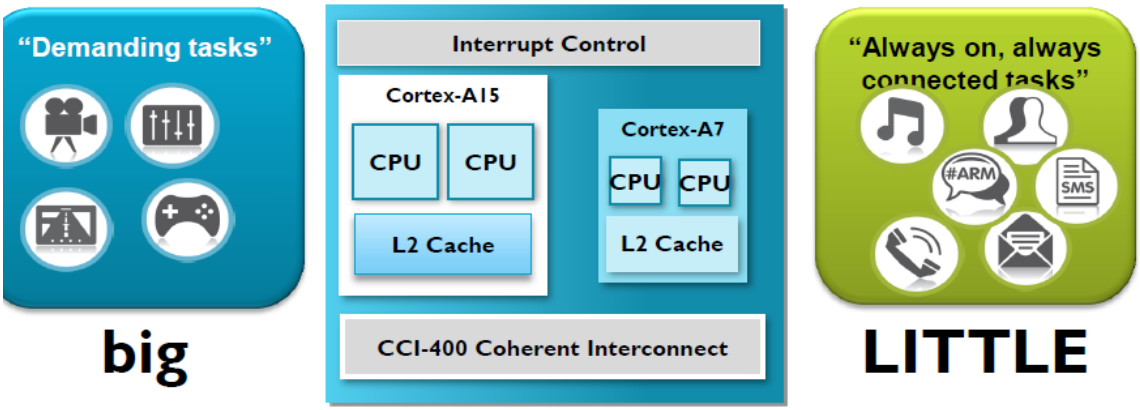

Figure 1.2. Example of monotonic core type consisting of big(OOO) core and little(InO) core [33]

.

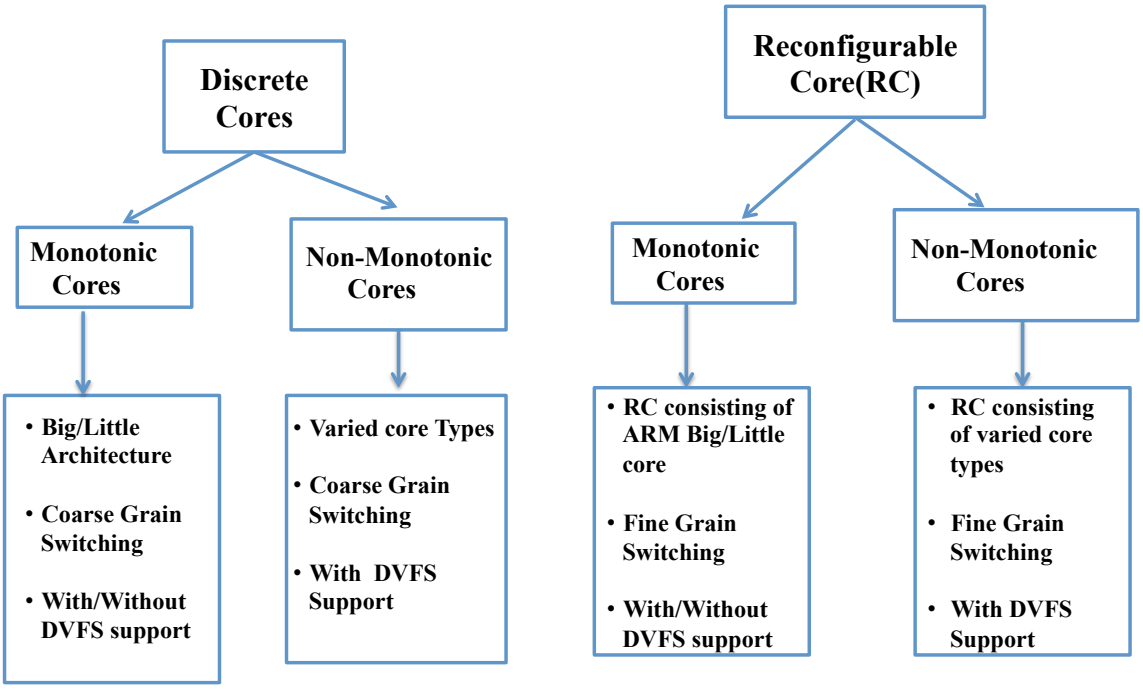

Figure 1.3. Overview of different architectures evaluated in this thesis for high performance and power efficiency.

mize power efficiency while minimizing the latency impact [52]. But thread swapping between core types in AMPs (monotonic/non-monotonic), can take thousands of cycles depending on the algorithm employed to swap threads and the mechanism to exchange contexts [81]. To amortize the large overhead associated with thread swapping, in most proposals, thread swapping decisions are made at a coarse instruction granularity of hundreds of thousands to millions of instructions [23, 52]. Therefore, achieving high power efficiency using AMP architecture requires: (i) a diverse set of cores that are optimized for various program phases, (ii) runtime analysis to determine the best core to run on, and (iii) low overhead of re-assigning a thread to a different core type.

Numerous opportunities to improve power efficiency at more fine grained instruction granularity [77, 103, 105] are missed out by a coarse grain approach. Existing static AMP architectures with multiple asymmetric cores and memory system incur large power/performance overhead for switching at fine granularity. To minimize thread switching overhead and obtain improved power efficiency of single threaded applications, this thesis explores reconfigurable/morphable architectures, where core resources are reconfigured on demand to suit the needs of the application. This supports heterogeneity within the core and minimizes thread switching overhead.

As the current industry standard AMP architecture is ARM's big/little, this thesis proposes a morphable/reconfigurable architecture design, where a single superscalar out-of-order (OOO) core can morph/reconfigure itself dynamically into an In-Order (InO) core at runtime. As the reconfiguration is performed within the same core and the application's architectural states are retained, the overheads of switching is reduced, thus enabling fine-grained switching between OOO and InO modes. Such a reconfigurable architecture can switch only between two monotonic core types. Having only 2 monotonic core types does not cater to the rapidly changing demands of all workloads thus leaving us with following un-answered questions.

1. How to design a reconfigurable architecture that adapts to varying demands of workloads at runtime with varying window sizes, execution bandwidth and voltage/frequency ?

2. How to design a run time mechanism for steering the application to the right core type ?

To address the above questions, we have performed a core design space exploration experiment to select a set of non-monotonic core architectures that are fundamentally different from the big/little architecture. The architectures of the cores can differ in fetch width, issue width, buffer sizes (e.g., Reorder buffer (ROB), Load Store Queue (LSQ) and Instruction Queue (IQ), clock frequency and operating voltage. We then use the selected core architectures to define the distinct core modes of the proposed reconfigurable architecture. Our reconfigurable core can dynamically morph into any one of these execution modes. Thus, this thesis proposes joint core resource resizing with dynamic voltage and frequency scaling (DVFS). This way, the reconfigurable core can mimic a high diversity asymmetric multi-core processor.

The computational resource requirements of an application change during its application execution and are not available beforehand. To take advantage of higher power efficiency opportunities, we need an on-line mechanism that is computationally fast and reasonably accurate in guiding the application to the right core mode at run time. The decision to choose what the best core to reconfigure into, needs to be taken at fine grain granularity. We propose a dynamic reconfiguration relying on online estimators to select the best core mode for the current needs of the executing application.

Many current processors employ DVFS aggressively to improve power efficiency and maximize performance [16, 20]. For example, memory bound phases of an application might not have sufficient ILP to keep the core busy and thus providing opportunities for scaling down voltage/frequency. Reducing voltage/frequency in such

	DVFS	AMP Design	Quantum
Coarse Grain	20-70 uSec	15-25 uSec	**10M Insts**
Fine Grain	80-100 nSec	200-300 nSec	**2K Insts**

Figure 1.4. Different AMP and DVFS designs made to switch at Fine/Coarse grain granularity (Quantum).

cases provides cubic reduction in power with minimal loss in performance [44]. Intel turbo-boost technology increases the frequency of active cores when other cores are idle, providing enhanced performance [20]. DVFS traditionally involves high overhead (tens of microseconds) as it depends on off-chip voltage regulator to switch from one voltage to another, thus allowing DVFS to be performed only at coarse grain OS switching granularity of millions of instructions [49]. Intel introduced fully integrated voltage regulator (FIVR) in their Haswell microprocessors, considerably reducing the voltage transition time, which enables switching from one voltage/frequency to another at fine granularity [17]. In this thesis, we study the trade off in power efficiency between applying only fine grain DVFS in static CMOS and reconfigurable architectures. We also study the effect of using on-chip and off-chip regulators for DVFS and compare to a reconfigurable architecture that uses an on-chip regulator. Figure 1.4, summarizes the overhead involved in switching between different AMP architectures and architectures that employ only DVFS. Reconfigurable architectures with fine grain switching have smaller switching overhead than coarse grain AMP architectures as shown in Figure 1.4. We analyze in this thesis each of the different architectures shown in Figure 1.4 and identify the architectures that provide the best performance/Watt. Thus this thesis would answer the following questions:

1. Can cores with dynamic heterogeneity consisting of DVFS and core reconfiguration online provide higher power efficiency than fine grain DVFS only?

2. How to design a run time scheme to switch between different core configuration and DVFS modes online? Does fine grain DVFS provide higher power efficiency than coarse grain DVFS? What are the power efficiency benefits of coarse grain core reconfiguration compared to fine grain core reconfiguration?

Recent literature has shown that reduced feature sizes and aggressive power management lead to increased process variation and soft error rates (SER) respectively [13, 24]. More severe defects at smaller technology nodes arise due to larger process variations, resulting in significant variation in characteristics of devices from what was intended at the design stage [15]. Soft errors occur when data stored in nodes of transistor are flipped due to radiation effects. The trend in soft error rates due to technology scaling is shown in Figure 1.5. Due to the increased soft error rate at lower technology nodes, we also discuss in this thesis the effects of soft-errors on core reconfiguration.

During core reconfiguration, there is trade-off between power efficiency and vulnerability to soft errors. For example, a workload that exhibits frequent cache misses achieves a higher power efficiency under lower voltage and frequency conditions. This leads to lower power without a decrease in performance as the performance bottleneck is the result of cache misses and not low frequency. Even though this may increase power efficiency, it also leads to greater vulnerability to soft-errors due to the lower voltage. Several studies report adverse impact of dynamic voltage and frequency scaling (DVFS) on SER [30, 88, 98, 106]. In this thesis, we explore dynamic core reconfiguration of non-monotonic core types with DVFS, for optimizing two objectives simultaneously: improving throughput/Watt efficiency and reducing vulnerability to soft errors.

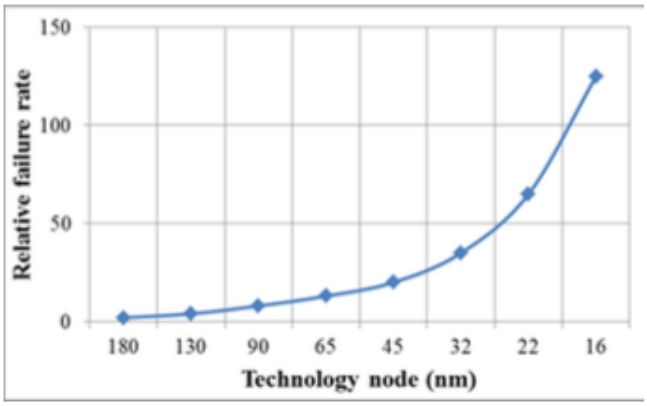

Figure 1.5. Trends in soft error rate [15]

In summary, we propose multiple solutions to improve power efficiency by bringing heterogeneity into the core and developing a fast run time mechanism to switch between different core modes at fine granularity. We also provide solutions to reduce SER combined with improving power efficiency at fine granularity, when switching between different core modes. We outline below the different sections in this thesis. In Chapter 2, we show our design of monotonic cores can improve power efficiency over existing monotonic core architectures. In Chapter 3, we introduce non-monotonic architectures and develop an on-line run time mechanism for mapping the application to the right core type. In Chapter 4, we present new non-monotonic architectures and show how they improve performance/watt when compared to designs described in Chapters 2 and 3. In Chapter 5, we compare our core design with various DVFS schemes. In Chapter 6, we present solutions to the problem of not only improving power efficiency but also reducing vulnerability to soft errors.

CHAPTER 2

IMPROVING POWER EFFICIENCY USING MORPHABLE ARCHITECTURE WITH MONOTONIC CORE TYPES

The computational needs of a program changes over time. Sometimes a program exhibits low instruction level parallelism (ILP), while at other times the inherent ILP may be higher; sometimes a program stalls due to a large number of cache misses, while at other times it may exhibit high cache throughput. Hence, the best core configuration (size of the queues, number of execution units etc.) for an application such that both energy consumption and performance are optimized, changes over time. In that spirit, Asymmetric Multicore Processors (AMPs) have been proposed to allow matching the computing needs of a thread to a core where it executes most efficiently [3, 4, 52, 79].

Monotonic AMPs employ two kinds of cores: out-of-order (OOO) big cores and in-order (InO) small cores. The big cores provide higher performance while the in-order small cores are more power efficient. As the benefits of such AMPs are highly dependent on a proper thread-to-core assignment, the threads are swapped between the cores at runtime so that the objective function (for example, performance or performance/power or energy) is improved for the current program phase. However, thread swapping incurs non-negligible costs. The swapping overhead can vary from a few thousand [4, 81] to millions of cycles [9, 51] depending on the algorithm employed to swap threads and the mechanism to exchange contexts. To amortize the large overhead associated with thread swapping, in most proposals, thread swapping decisions are made at the granularity of hundreds of thousands to millions of instruc-

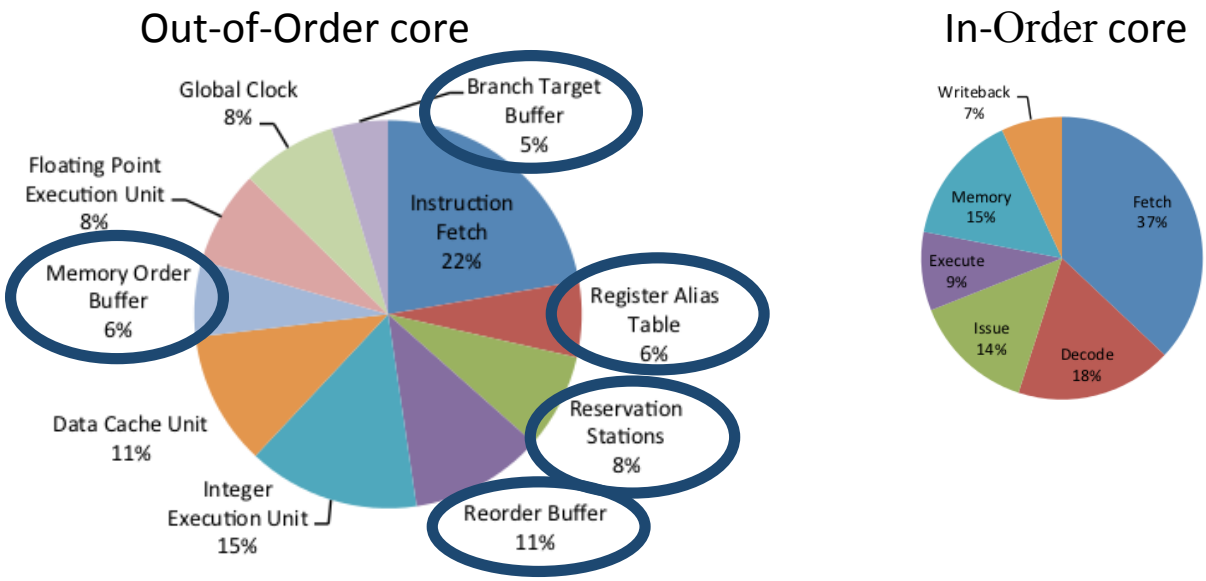

Figure 2.1. Energy consumption difference between OOO and InO core for SPEC benchmarks

.

tions [9, 51]. Unfortunately, numerous opportunities to improve performance/power at a more fine grained instruction granularity [77, 103, 105] are missed out by such approaches. Therefore, there is need for a mechanism to realize these opportunities without incurring large thread swapping penalties.

OOO core relies heavily on speculative execution by making use of data structures such as ROB and reservation stations to support OOO execution. Data movement between these structures consume significant power as shown in Figure 2.1. In-order core consumes significantly less power as it does not execute instructions speculatively and thus it does require fewer hardware structures compared to an OOO core. Figure 2.2 shows the IPC difference resulting from running the workload *mcf* on the OOO and InO cores. In this figure, the IPC is sampled at a coarse grain instruction granularity of 50K instructions. Here, it can be seen that at no point is the IPC of the InO core comparable to that of the OOO core. Figure 2.3 compares the IPC difference when running the workload *mcf* on the OOO and InO cores at a fine grain granularity

of 500 instructions. It can be seen that, not only are the IPCs of the two cores comparable in some of the instruction intervals, but at some points in the plot, the InO core outperforms the OOO core. The InO core with simpler pipeline structures experiences less stalls when compared to an OOO core for benchmarks that experience significant memory misses. The InO is the power efficient core and from the figure, it is clear, that at smaller instruction granularities, when the performance difference between the two core types is small, we can gain in throughput/Watt by switching from the OOO to the InO core. However, swapping threads at such a small granularity in current AMPs, will likely negate all benefits. Hence, there is need for a more fine grain switching mechanism that does not incur large thread swapping penalties. In this chapter we address the following questions:

1. How to design an AMP architecture that can allow switching between AMP core types at fine grain instruction granularity?

2. How to design a simple runtime scheme that determines when to switch between AMP core types?

3. How fine grain should core switching be done to achieve the highest throughput/Watt

To address the above issues, we propose a core morphing mechanism that reaps most of the benefits of AMPs, without incurring the penalty associated with thread swapping. Our proposed mechanism introduces heterogeneity within a single core by morphing it from OOO to InO core and vice-versa. Certain Intel processors feature a special debug mode in which the OOO core turns into an InO core [51]. We extend this mechanism for improving power efficiency by opportunistically switching to the InO mode, if deemed beneficial. As the morphing is performed within the same core and the architectural states are retained, the overheads associated with our scheme are negligible, thus enabling fine-grained switching between OOO and InO modes

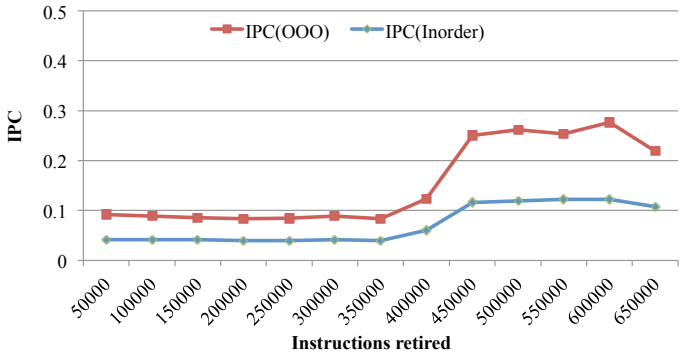

Figure 2.2. IPC comparison between OOO and InO cores when executing the workload *mcf* at coarse grain instruction granularity of 50K retired instructions.

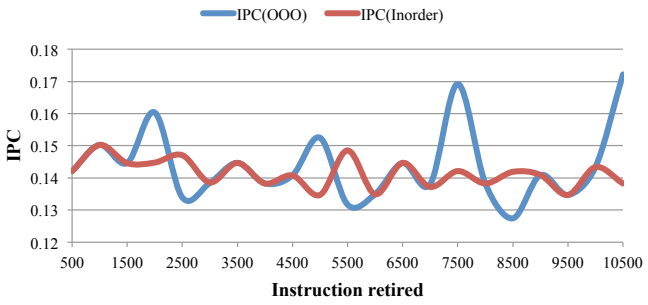

Figure 2.3. IPC comparison between OOO and InO cores when executing the workload *mcf* at fine grain instruction granularity of 500 retired instructions.

2.1 Related Work

We now provide an overview of recent advances made in literature that closely relate to our proposed scheme.

2.1.1 Asymmetric Multicore Processors (AMP)

A number of proposals have been made on the topic of AMPs for performance and performance/Watt gains. Kumar *et al.* in [54] make use of cores of different sizes to best match the thread to one of the available cores. However, only a single thread was run on a system consisting of four cores. This greatly simplifies the scheduling problem in AMPs. AMPs have also been used to eliminate serial bottlenecks in parallel workloads. Suleman *et al.* [95] considered such an architecture consisting of big cores and several small cores. The big cores were used as accelerators. The central component of every proposal focusing on AMPs revolves around the mechanism to determine the best thread to core assignment such that while switching to a more power efficient core, the power hungry structures in the high performance core are not utilized. Several researchers have also proposed AMP core types for operating system codes [62] and ILP and MLP intensive codes [73].

2.1.2 Morphable or Dynamic Multicores

There have been several proposals that advocate dynamic morphing of multicores or single cores such that performance and power efficiency are enhanced at run time. In a number of proposals, the starting point is a multicore consisting of small cores which then fuse together into a large OOO core on demand [48, 75, 96]. Such approaches suffer from additional latencies that arise from combining resources from various cores. A different scheme was adopted by Khubaib *et al.* in [47] where they start with a baseline OOO core that morphs itself into a simultaneously multithreaded InO core depending on the number of incoming threads. All such schemes

require significant changes to the microarchitecture making them difficult to adopt in practice.

Dynamic sharing of processor resources for power and performance benefits is also a well explored area. Kumar *et al.* [53] explore sharing of various large structures in the multicore for power and area savings. In [81], Rodrigues *et al.* explored dynamic exchange of execution units such that performance/Watt is improved. All such schemes require extra circuitry that must be designed and verified. In [61], Lukefahr *et al.* make a proposal that is similar to ours. In their scheme, heterogeneity is introduced into a single core by provisioning two execution backends to the core. One backend is an OOO while the other is InO. Both backends share the caches and fetch units. The difference between this scheme and ours is explained in detail in the proposed architecture section later in this chapter.

2.1.3 Recent advances made in thread to core Mapping in AMPs

Prior knowledge about the computational resource requirements of different applications is generally not available beforehand. Hence, there is a need for an online mechanism to characterize the time-varying program behavior and determine the appropriate mode (OOO or InO) at runtime such that the throughput/Watt of the executing application is maximized. We next cover some of the recent advances made in scheduling in AMPs.

2.1.3.1 Sampling Based Techniques

Online learning schemes offer a more practical solution to the AMP scheduling problem. These schemes learn the characteristics of the workloads online and based on this make informed thread scheduling decisions. Online learning schemes primarily employ phase classification and sampling techniques to perform scheduling [9, 52, 81, 102]. Whenever a stable phase change is detected, the new phase is sampled on all the core-types in the AMP [81, 84]. Winter *et al.* explored different techniques such as

brute force, greedy and local search for thread to core mapping using heterogeneous cores [102]. For implementing each of these techniques, they sample the thread on each of the cores to make thread switching decisions. Becchi *et al.* proposed a steering algorithm by sampling thread on both fast and slow cores and computing the speedup factor for deciding on thread switch [9]. Such sampling based schemes pose significant overhead as the application must be sampled on each of the core types before migrating the thread to the preferred core. Thus, such schemes may not be scalable for a many-core system [23] and can only be employed at coarse grain instruction granularities and as such cannot be used for our purpose.

2.1.3.2 Heuristics Based Techniques

Heuristics based thread to core mapping approaches are improvement over sampling based schemes as they eliminate the overhead involved in sampling. Prior works have used certain metrics such as L2 miss rate or stall information to determine the right thread to core assignment [51, 83]. Saez *et al.* proposed a steering algorithm for monotonic core type that relies on estimating the L2 miss rate [83]. It is, however, unclear whether using L2 misses alone is sufficient to make thread to core assignment decisions such that performance/power is maximized. Koufaty *et al.* determine thread to core mapping in an AMP, using program to core bias which is estimated online using the number of external and internal stalls [51]. Their objective was only to improve performance. Patsilaras *et al.* determined the amount of MLP using l2 miss statistics to determine the right thread to core assignment in AMPs [73].

2.1.3.3 Estimation Based Techniques

Estimation-based techniques are closest to what is considered in our work. Numerous schemes that employ regression-based analysis techniques [21, 45, 85] for thread to core mapping in AMPs have been published. In these, regression is used to estimate power and performance in the same core. Other works estimate the performance

and/or power of running a thread on another core in the AMP using statistics such as cache misses and pipeline stalls gathered on the host core [23, 51, 61, 80, 93]. In our work, we employ performance monitoring counters (PMCs) to estimate the IPS^2/Watt of the thread in both modes (OOO and InO) using the PMC of the host core. Based on this estimation, the core that is expected to provide a higher IPS^2/Watt is chosen.

2.2 Proposed Architecture

In our scheme, only a single core is considered. In the baseline mode, the core operates in the OOO mode providing high performance. However, during low IPC phases, the core may be morphed into the InO mode for a higher performance/Watt. A similar switch is made from InO to OOO when these benefits are predicted to have diminished. By switching between operation modes, the proposed scheme takes advantage of heterogeneity while incurring minimal overhead upon a mode switch.

Figure 2.4 shows the considered baseline core which is a 4-way issue OOO superscalar core. The backend of the baseline core is provisioned with register alias table (RAT), load/store queue (LSQ) and Re-Order Buffer (ROB) to facilitate OOO execution and inorder commit. During high-ILP program phases, a significant performance improvement is achieved by executing the thread on the OOO baseline core. However, when the processor is waiting for long-latency memory operations to complete or experiences stalls due to data dependencies, most of the core resources are idle wasting static power. For such low-IPC phases, a low-power InO core may be more power efficient. The OOO mode relies heavily on speculative execution by making use of data structures such as the ROB and the reservation stations to ensure OOO execution but in-order commit. Data movements between these structures consume significant power. For some phases of a program, this increase may not commensurate with the performance improvement resulting in poor throughput/Watt. It can be seen in Figure 2.4, that the issue and execution stage power for the OOO mode are significantly

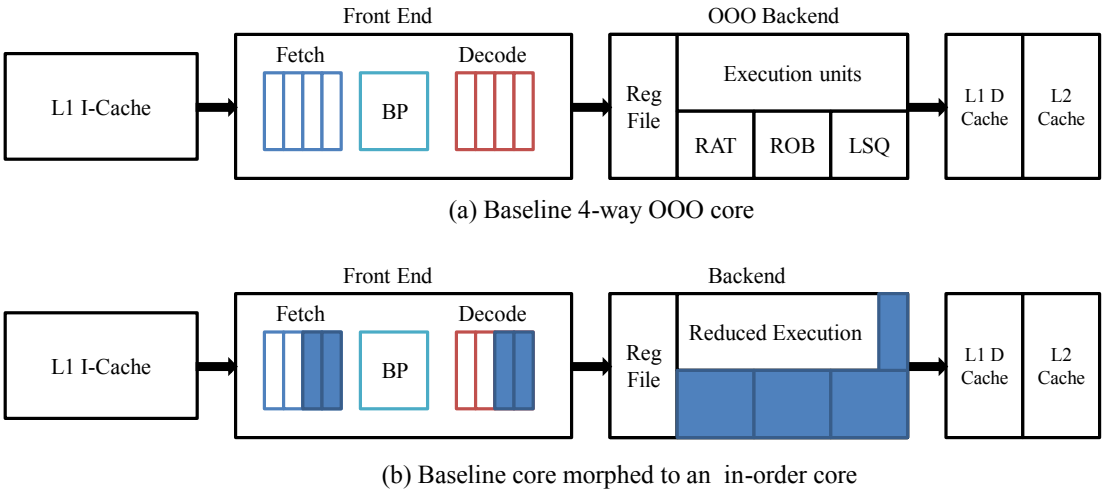

Figure 2.4. (a) High-level view of the 4-way OOO baseline core. (b) The InO core obtained by reconfiguring the baseline core. The shaded regions indicate the units that are power-gated during InO execution. BP - Branch Predictor.

higher than the InO mode. These are the stages where the data structures are used and accessed the most. This result shows that the power expended in the OOO mode can be significantly higher than in the InO mode. When such increase in power is not accompanied by a significant performance gain, a switch in mode from OOO to InO may be beneficial. To this end, during low-ILP/memory intensive phases, we power off the ROB, RAT, and LSQ, enabling only in-order execution/commit. Thus, the baseline OOO core is opportunistically morphed into an InO core providing significant power benefits. In this mode, the baseline core supports only in-order execution and retirement of instructions. As the performance of the core in InO mode is expected to be low, we reduce the fetch width of the core from 4 to 2, and further more, power off half of the decoders and, shut-down few of the multiple execution units. While in InO mode, if the program moves to a high-ILP phase, the shut down units are powered on, reverting back to the baseline OOO execution.

Our proposed core morphing scheme is similar to the one proposed by Lukefahr *et al.* [61] but differs in the following ways. Firstly, Lukefahr *et al.* employ two different backend pipelines and decode units while our scheme uses the same for both modes

(OOO and InO). The additional units increase the core area and design/verification effort. More importantly, the scheme proposed in [61] requires the architectural states to be transferred across the two pipelines which adds to the overhead. In contrast, the same register file is used by the two modes in our scheme. Finally, our scheme differs in when the mode switch (OOO to InO and vice versa) actually happens. Whenever the scheme decides to switch from OOO to InO mode, the units are power gated, we then drain the pipeline and the subsequent instructions are re-fetched in InO mode. When switching from InO to OOO mode, the units are powered back on and, the head and tail pointers of the ROB are re-initialized to point to the same slot. Thus, the ROB is presumed to be completely empty when the core is morphed back to the baseline OOO mode. The fact that we make use of existing facilities in the processor core to enable reconfiguring makes our proposal more practical.

2.3 Runtime Reconfiguration Management

Reconfiguring from the OOO to the InO mode of operation needs to be done at runtime. This requires a mechanism that makes dynamic decisions depending on the characteristics of the currently executing workload. The decision metric chosen for selecting new mode is based on computing IPS^2/Watt on-line [36, 5]. The metric IPS^2/Watt gives higher weightage to performance than power. Our proposed core reconfiguring scheme accomplishes this task by estimating the IPS^2/Watt of the current execution phase of the application in both the modes (OOO and InO) as explained next.

2.3.1 Power and Performance prediction mechanism

The current characteristics of the application being executed on a core can reveal considerable information about how suitable the core is to that application. For example, an application phase that results in a significant number of misses in the

level-1 cache will result in low performance and high power consumption in OOO core. Executing this phase on an InO core would make more sense with respect to IPS^2/Watt. In order to assess the current characteristics of the application being executed, we make use of Performance Monitoring Counters (PMC).

In order to estimate the IPS^2/Watt, both performance and power need to be measured or estimated. Performance measurement is straightforward, while real time power or energy measurement is not. PMCs have been used as a proxy to estimate power in the past [22, 87] and we follow a similar approach. Note that most previous work make use of PMCs to estimate power on the same core while we need to estimate power and performance on the currently active mode (OOO/InO), as well as the other mode (InO/OOO) to make an informed decision.

2.3.1.1 PMCs explored in this study

There are many events that take place in a modern processor but some of them provide better hints than others about the performance and power of the currently executing application. To this end, we have explored fourteen different performance counters. We considered (i) the number of retired instructions of each type (integer, floating-point etc.), (ii) memory hit and miss counters (level-1, level-2 and TLB misses), (iii) number of mis-predicted and correctly predicted branch instructions, (iv) number of instructions fetched and instructions retired per cycle (IPC), and (v) pipeline stall counters which include stalls resulting from lack of reservation stations, load/store queue entries, RAT and ROB slots.

2.3.1.2 Shortlisting the PMCs

In general, we expect a higher estimation accuracy using a large number of counters. However, there is a limit on the number of counters that may be accessed at the same time. This limit varies from one architecture to another. For example, in the Intel XScale processor [22], only two counters may be accessed while for the

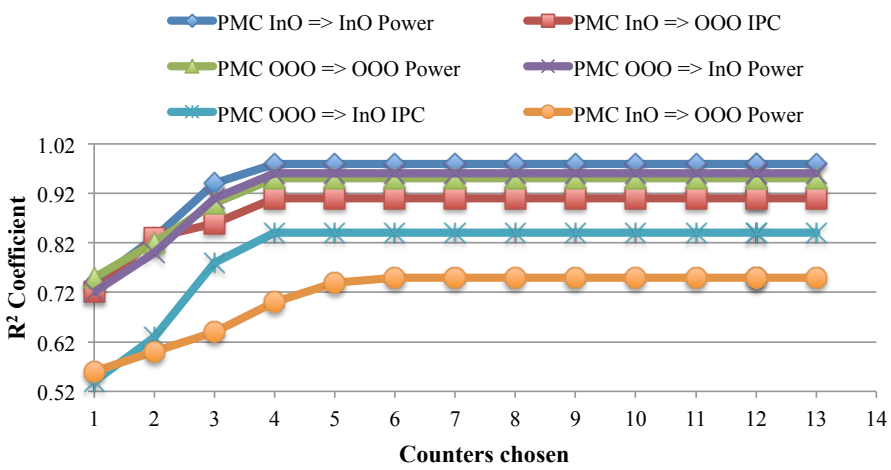

Figure 2.5. R^2 coefficient as a function of the number of chosen PMCs. PMC InO => Power OOO denotes that using the performance counters of the InO mode, we estimate the power on the OOO mode.

AMD Phenom processor, at most five counters may be accessed at the same time [87]. There is, therefore, a need to find a minimal subset of PMCs that have the highest correlation with power and performance both in the currently active mode, and the other.

To accomplish the task of making the right choice of PMCs, we devised a greedy heurestic that searches the counter space iteratively. During each iteration, our counter selection algorithm picks a new counter that best fits the estimating parameter (performance or power) along with the set of counters already chosen in previous iterations. We used linear models for curve-fitting and the best fit is qualified by the R^2 correlation coefficient. During the initial few iterations, the value of the R^2 coefficient increases steeply as more counters are added, but it tends to saturate later. The best set of counters is around the region where the R^2 coefficient tends to saturate. In order to carry out such an analysis, we performed regression analysis using the PMCs as variables and the performance and power as objectives.

As expected, increasing the number of counters yields better R^2. However, we arrive to the point of diminishing returns after certain number of counters. The

Table 2.1. Power and performance estimation of the other mode using the performance counters' values in the current mode. *L1h - L1 Hit, Bmp- branch miss prediction, S - Store, L- Load, DS- Dispatch Stall*

Estimating Parameter	Expression
InO ⇒ OOO IPC	$4.5 \times 10^{-3} \times$ L1h + 4.417 × IPC - 0.0273 × Bmp - 2.3255
InO ⇒ OOO Power	0.080 × L1h + 71.15 × IPC - 0.4112 × Bmp - 38.46
InO ⇒ InO Power	0.0047 × L1h + 13.062 × IPC - 0.0069 × S - $7.4 \times 10^{-5} \times$ DS + 1.5547
OOO ⇒ InO IPC	- 0.00616 × L1h + 0.06671 × IPC - $4.2 \times 10^{-4} \times$ Bmp - $7.5 \times 10^{-5} \times$ DS + 0.2768
OOO ⇒ InO Power	-0.0039 × L1h+ 0.9022 × IPC + 0.0104 × S - 0.0103 × Bmp + 4.4669
OOO ⇒ OOO Power	0.0141 × L1h + 13.81 × IPC + 0.0295 × S - 0.0118 × Bmp - 0.2989

resulting number of selected counters and the R^2 value obtained in each of the 6 estimation (Power/IPC) are shown in Figure 2.5. In Figure 2.5, PMC InO => OOO IPC denotes using the PMCs of InO mode to predict IPC in the OOO mode. The average R^2 value across all modes is 0.85, showing high accuracy in power/performance estimation. The final expressions obtained are shown in Table 2.1.

The average error observed when using PMCs on one mode (OOO/InO) to predict power in that mode as well as performance and power on the other mode (InO/OOO) is show in Figure 2.7. While estimating the OOO parameters (IPC and power) for the InO mode using PMCs in the InO mode, the average % error in estimating IPC and power is around 16% and 10%, respectively. Similarly the average % error in computing the InO parameters from OOO core was found to be 15% and 8%, respectively. Error in estimating power using counters in the same mode was found to be around 9% for the OOO mode and 8% for the InO mode. Using the estimated power and performance values, IPS^2/Watt for both modes is then computed using PMCs from the currently operating mode. Although the average estimation error is reasonably low, the actual estimation error may be considerably higher at some time instances and this may cause wrong morphing decisions. Therefore, we analyzed

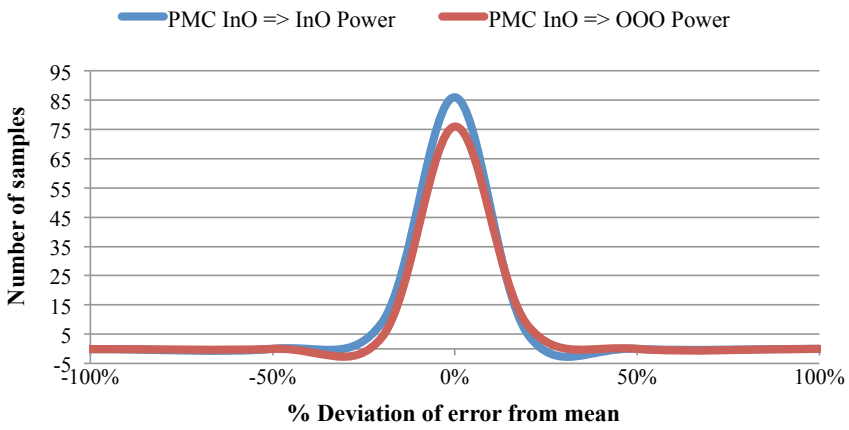

Figure 2.6. Distribution of estimation error when using PMCs of InO mode to estimate power in InO and OOO modes.

the temporal distribution of errors and the results are shown in Figure 5.1. This figure depicts the error in estimating power in InO and OOO modes using PMC of InO mode. We observe that the deviation of the errors from the mean is low for the majority of sample points with up to 75% of the sample points lie between + or -10% from the mean. This demonstrates that the average error is a sufficiently good indicator for the instantaneous estimation error. In our experiments we have observed very few decision errors.

2.3.1.3 Capturing Application Phase behavior

To adapt to the computational needs of program, it is important to identify the program phase behavior and find out the affinity of the program phase to a specific core mode. Morphing from OOO to InO mode or vice versa should be considered only when the application has entered a stable phase behavior or else the overhead of scheme to morphing will become prohibitive. After a certain number of retired instructions, referred to as *window*, a tentative morphing decision about the best mode (OOO or InO) is made based on the IPS^2/Watt estimations. To avoid too frequent switching between the modes (InO and OOO), we prefer to wait until the

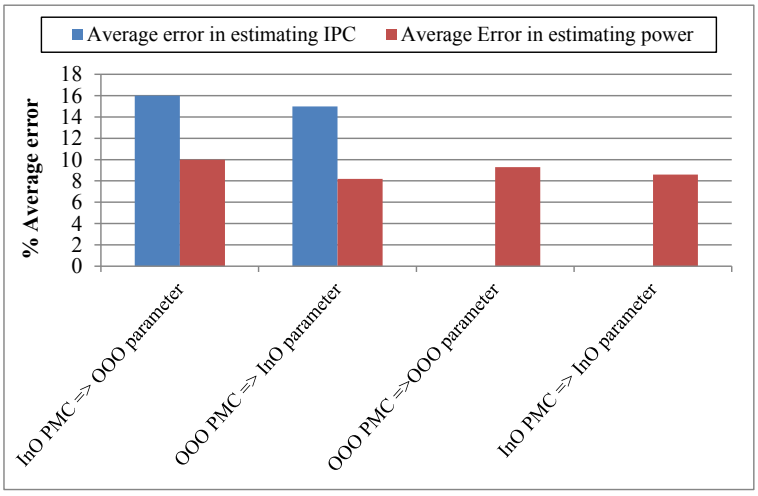

Figure 2.7. Average error (in %) observed in estimating IPC and power of OOO (InO) mode using InO (OOO) counters.

new execution phase of the thread has stabilized. W denote by n the total number of retired instructions during this period where $n = \textit{history depth} \times$ window length. For example, if for the past n committed instructions, moving from OOO to InO mode was the most frequent decision, it may be predicted that the application has entered a phase where InO mode may provide higher IPS^2/Watt. The window size and history depth need to be determined experimentally. We have conducted a sensitivity study to quantify the impact of window length and history depth on the achieved benefits. The window size and history depth combination that yields the highest IPS^2/Watt for the entire program execution would be the best choice.

The window length was varied from 250 to 1000 instructions. Within a particular window, the history depth (n) was varied from 2 to 8 in steps of 1. For example, a history depth (n) of 4 and window length of 500 indicates that we make a reconfiguration decision at the end of every 2000 instructions. To determine the optimum window size and the history depth, we ran a set of 10 benchmarks. After each benchmark was run for 1 billion instructions (after skipping the initial 2 billion instructions), we computed the average increase in IPS^2/Watt of the proposed scheme (that can

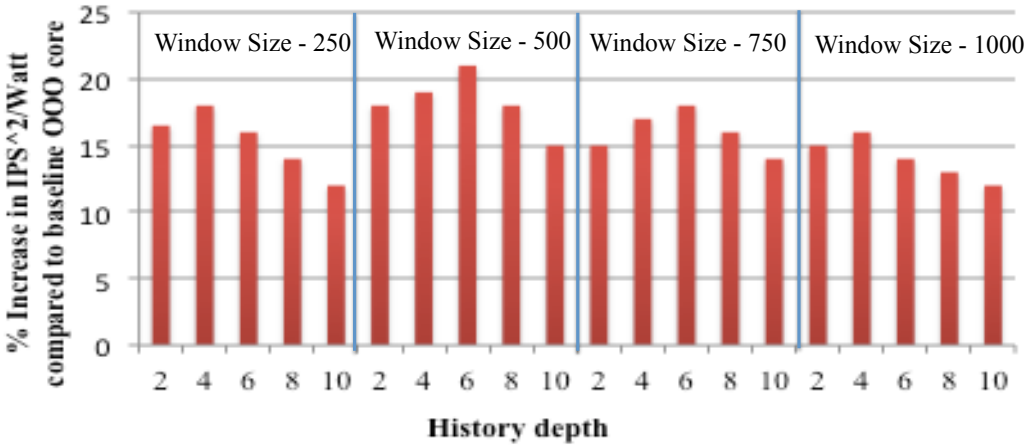

Figure 2.8. % Average increase in IPS^2/Watt of the proposed scheme w.r.t the baseline OOO core for different values of window length and history depth.

switch between OOO and InO modes) over the baseline OOO core. The decision to switch between operation modes is determined by the most frequent decision made within the history depth. As shown in Figure 2.8, window length of 500 and history depth of 6 provides the maximum increase in IPS^2/Watt. The above computation of IPS^2/Watt increase takes into account the overhead for switching between modes, as explained in the next section. Thus, in all our future experiments, the window length of 500 and history depth of 6 is used.

2.3.1.4 Switching between OOO and InO modes

Due to the low overhead associated with our morphing scheme, we dynamically morph from one mode to another at a fine-grained instruction granularity. As mentioned earlier, InO mode with reduced architectural units provides better power efficiency at the cost of performance. It is critical that we move into the InO mode only when we expect lower power consumption without compromising performance significantly.

At the end of every n commited instructions, we decide to move to InO mode only if the expected IPS^2/Watt in InO mode is greater than that of the OOO mode by a defined threshold. Switching threshold parameters are discussed in Section 2.4.1

2.3.1.5 Reconfiguration overheads

Previously proposed schemes for morphing [61, 79] or swapping of threads between asymmetric cores [4, 9, 52, 80] incur large overhead and as a result, thread swapping or morphing were done at a very coarse grain granularity. The overheads for these schemes arise from the transfer of architectural state requiring a warm up the cache and the branch predictor [3, 80] or due to a high communication latency to send or receive data operands [79]. In our proposed scheme, morphing is done within the core and thus it avoids all the above overheads as there is no need to change the state of the register file, caches and branch predictors. The overhead associated with our scheme is due to the power gating/power up of the ROB, RAT and LSQ units and partial power on/off of fetch, decode and execution units while switching between OOO and modes. When power-gating individual units, there is no dynamic power consumed and the static energy consumed by these idle units is not very high providing us with increased power savings. Power gating/power-on of all the blocks simultaneously may lead to a sudden power surge and thus we employ staggered power gating where one block is gated every clock cycle. To compute IPS^2/Watt, seven expressions (shown in Table 2.1) must be evaluated online, which require four multiply and accumulate (MAC) operations per expression. The resulting computation overhead is about 30 clock cycles. Average overhead for pipeline drain is estimated to be 60 cycles. Detailed information on the reconfiguration controller which contains MAC unit is explained in next chapter. Thus, the average overhead when switching between modes is conservatively assumed to be 100 clock cycles for every switch. Actual switching overhead is calculated in runtime by taking into account the draining of resources from buffers before mode switch can be initiated. The switch between OOO and InO modes is handled in hardware and no changes are required to the operating system.

Table 2.2. Baseline OOO core parameters considered. The values in parenthesis represent the change while in InO mode.

Param	Value	Param	Value
Issue	4 (2)	INTREG	96 (NA)
FPREG	80 (NA)	INTISQ	36 (36)
FPISQ	36 (NA)	LS units	3 (1)
LSQ	128 (NA)	ROB	128 (NA)
L1(I/D)	64K	L2	2M
Freq (GHz)	1.6	Type	OOO (InO)

Table 2.3. Execution unit specifications for the baseline core. (P - Pipelined, NP - Not pipelined, PP - Partially pipelined).The values within parenthesis represent the change while in InO mode

FP DIV	FP MUL	FP ALU
1 unit, 21 cyc, P	1 unit, 5 cyc, P	2 (1) units, 3 cyc, P
INT DIV	INT MUL	INT ALU
1 unit, 23 cyc, P	1 unit, 8 cyc, P	4 (2) units, 1 cyc, P

2.4 Results and Analysis

In this section, we evaluate our proposed core morphing scheme. The core parameters considered in this work are listed in Tables 2.2 and 2.3. Most of these parameters were take from [32]. As shown in Table 2.2 , the OOO core is provisioned with large resources (e.g., integer and floating-point registers, issue queues and L2 cache) which is representative of modern super-scalar processors. The changes to the architectural parameters and the execution units in the InO mode are shown in parenthesis in Tables 2.2 and 2.3 , respectively. We used Gem5 as our cycle accurate simulator with integrated McPAT modeling framework to compute the power of the core and L1 caches [10, 59]. The evaluation was carried out using SPEC2006 and SPEC2000 benchmarks suites [11, 89]. Each of the benchmarks were run for 1 billion instructions after skipping the first 2 billion instructions.

2.4.1 Power Efficiency

The decision to switch from one configuration mode to another is based on the IPS^2/Watt threshold. We now explain the process of determining the IPS^2/Watt

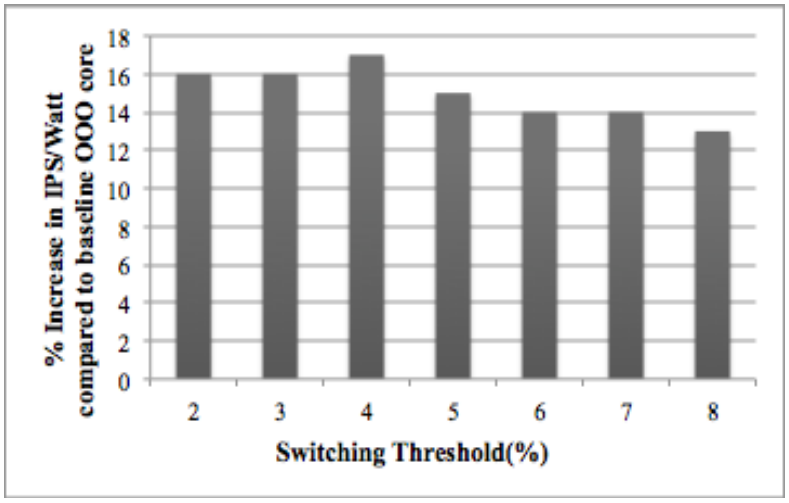

Figure 2.9. % Increase in IPS/Watt vs various switching threshold

threshold. As shown in Figure 2.9, IPS^2/Watt threshold was varied from 2% to 8% for the determined window length of 500 and history depth of 6. We observe that at smaller threshold of 2%, the achieved improvement in IPS/Watt is 16%. As we increase the threshold to 4%, we observe a higher improvement in IPS/Watt. At the lower threshold (2%), reconfiguration can happen too frequently thus increasing the reconfiguration overhead resulting in reduced IPS/Watt. Beyond 5%, there is smaller benefit due to reduction in the number of reconfiguration. The IPS^2/Watt threshold was therefore set to 4%.

Figure 2.10 shows the increase in IPS/Watt when compared to the baseline OOO core. Memory intensive benchmarks such as *soplex* and *mcf* provide an IPS/Watt improvements of 37% and 38%, respectively. Higher IPS/Watt improvement is obtained, since these benchmarks stall the pipeline frequently on memory misses and running them on InO core is more power efficient. Branch intensive benchmarks such as *astar*, *sjeng* and *gobmk* also achieve high IPS/Watt by morphing into InO core during periods of high miss-prediction activities. Benchmarks which are highly compute intensive such as *bzip2*, *h264ref* and *apsi* do not incur many memory stalls and thus do not morph into InO core frequently, resulting in lower IPS/Watt improve-

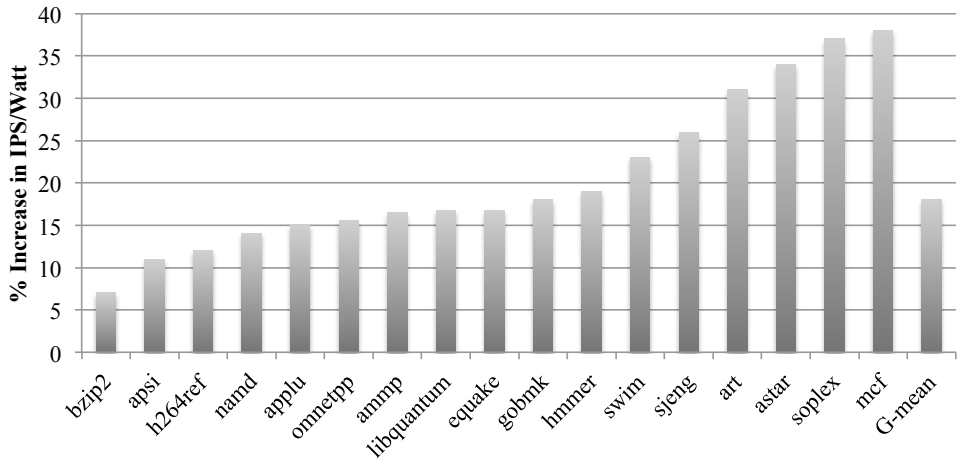

Figure 2.10. % Increase in IPS/Watt of proposed scheme w.r.t the baseline OOO core.

ments. Figure 2.11 compares the average IPS^2/Watt, IPS/Watt and energy savings compared to the baseline OOO core. IPS/Watt improvement of 17%, energy savings of 19% and IPS^2/Watt improvement of 21% are obtained.

2.4.2 Comparison to other Switching schemes

We compare our PMC-based fine grain morphing scheme referred as *FineGrain_PMC* to three other switching schemes, namely: (i) Sampling based switching within a morphable architecture, referred to as *CoarseGrain_sampling*; (ii) Oracular scheme referred to as *Oracular*; and (iii) PMC-based switching at coarse grain granularity, referred to as *CoarseGrain_PMC*. Traditional AMP architecture such as big.LITTLE allow switching to be done at a coarse grain instruction granularity of about hundreds of millions of instructions which is at the granularity of phase change. These architectures employ sampling based techniques to determine the right AMP core type. As a result, they cannot exploit low performance phases that exist at a finer granularity due to the high overhead involved in sampling based scheme. We compare our traditional coarse grain sampling (*CoarseGrain_sampling*) based switching in AMP with our *FineGrain_PMC*. To model the sampling based scheme we use

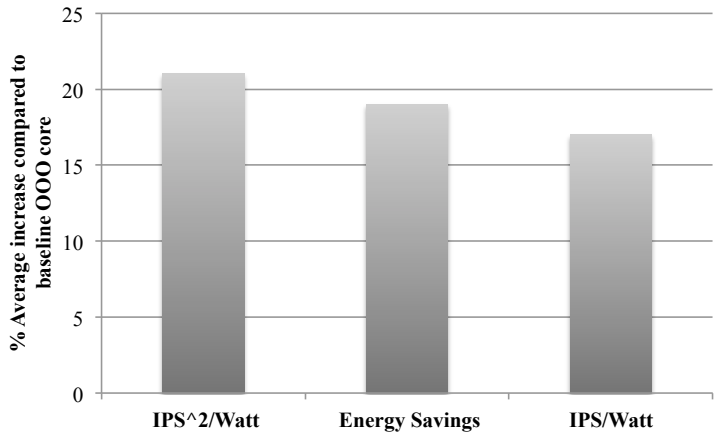

Figure 2.11. % Increase in IPS^2/Watt, IPS/Watt and energy savings of proposed scheme w.r.t the baseline OOO core.

two parameters, the switching interval and sampling interval. To make the decision to morph into a different core type, after every switching interval the application is sampled on each of the core types. The best core type which is found during the sampling interval is the core we morph into, where the application is run for the next switching interval. The switching interval is taken to be 1M with sampling interval of 10K instructions. In the oracular scheme, an oracle steering algorithm is used to guide the core in morphing decisions. Switching between core modes is performed at instruction granularity of 3K as determined earlier. Thus, for every 3K instructions retired, the oracular scheme chooses the core that will best suit the application in the next 3K interval.

As shown in 2.12, the oracular scheme provides an IPS/Watt improvement of 31%. This scheme provides the upper-bound for maximum IPS/Watt that could be achieved by our scheme. *FineGrain_PMC* scheme achieves a higher by 6% IPS/Watt compared to *CoarseGrain PMC* scheme. *CoarseGrain_sampling* scheme provides a lower by 10% IPS/Watt compared to the *FineGrain_PMC* scheme. Sampling is performed at a coarse grain level missing opportunities available at lower granularity.

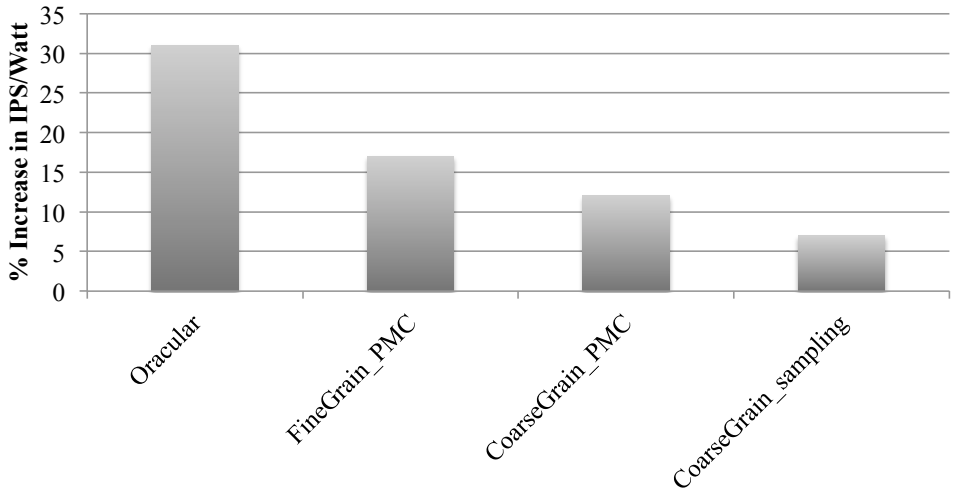

Figure 2.12. Comparison of four switching schemes.

The number of mode switches for the chosen window length and history depth is shown in Table 2.4 for all the benchmarks. As expected, benchmarks which achieve increased power savings exhibit higher number of switches into InO mode. Table 2.4 also shows the percentage of time spent by each benchmark in the InO mode. Benchmarks like *mcf* and *soplex* spent more than 50% of time in InO core thus providing improved power efficiency.

Figure 2.13 shows the IPS/Watt improvement with increasing values of core reconfiguration overhead. Our initial estimated cost of overhead for reconfiguration is 100 cycles, the actual overhead is computed during simulation taking into account draining of banked resources. From Figure 2.13, we can observe that as the overhead increases from 100 to 500 cycles, IPS/Watt decreases by 2%. On overhead of 1K cycles and beyond, there is larger decrease in IPS/Watt indicating that switching at fine granularity must have a fast switching mechanism.

We also compare our reconfigurable design with architectures that resemble ARM Big/Little architecture [35]. ARM Big/Little architectures consist of high performance OOO 'Big' core and energy efficient InO 'Little' core. It differs from our

Table 2.4. Number of switches per million instructions and percentage time spent by benchmarks in the InO mode

Benchmark	Switches/million instruction	Percentage time spent in InO mode
bzip2	64	12
apsi	70	14
h264ref	75	15
namd	90	18
applu	94	19
omnetpp	110	22
ammp	125	23
libquantum	133	26.6
equake	140	28
gobmk	148	29
hmmer	153	30.6
swim	155	31
sjeng	162	42
art	165	48
astar	170	50
soplex	183	55
mcf	185	60

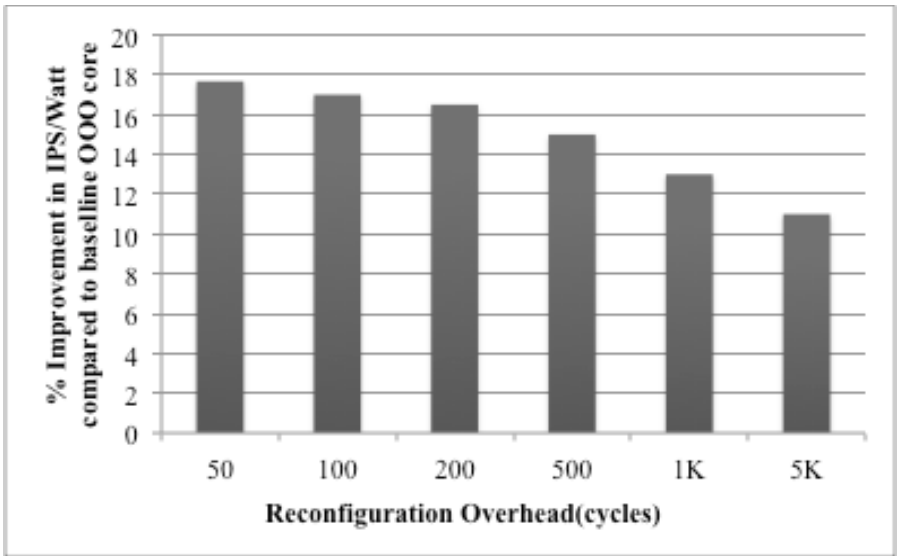

Figure 2.13. Impact of core reconfiguration overhead on IPS/Watt improvement

morphable architecture, where each core has two separate core modes. The core parameters are the same as those shown in Table 2.2. The PMC based online scheme is used to switch between core types. Switching between core types involves overhead of 20 usec [35]. Due to higher overhead of switching, migration of applications between cores is done at coarse granularity of 10M instructions. We then compare the throughput/Watt obtained using a reconfigurable design against the ARM Big/Little architecture. We find that, the reconfigurable design with fine grain switching achieves a higher, by 7% throughput/Watt compared to the ARM Big/Little architecture.

2.5 Conclusion

Applications experience a change in characteristics over time. Hence, a different core configuration (size of the ROB, number of execution units etc.) may be more suitable with respect to energy and performance at different time instants. Therefore, AMPs have been considered to support the diverse needs of applications. Here, depending on the current application characteristics, threads are swapped between the available cores in the AMP such that the objective function (for example, energy or performance) is optimized. Prohibitive thread migration overheads limit the instruction granularity at which such thread swapping decisions may be made, even though many opportunities present themselves at fine grain granularities. In this chapter, we have considered an architecture that is capable of realizing these benefits. Here, depending on application characteristics, a super-scalar OOO processor may morph itself into an in-order (InO) core at runtime, if deemed to be beneficial. Such morphing is feasible as it resembles the existing debug feature present in certain Intel processors. The decision to morph between operation modes (OOO/InO) is made using information gathered from performance monitoring counters. The proposed scheme opportunistically morphs into InO mode to maximize IPS^2/Watt. Our re-

sults indicate that, on average, an IPS/Watt improvement of 17% is obtained over the baseline OOO core.

CHAPTER 3

IMPROVING POWER EFFICIENCY OF NON-MONOTONIC PROCESSORS VIA PROGRAM PHASE CLASSIFICATION

Most commercial AMP architectures are monotonic ones, that include either high performance/high power and low performance/low power core types. Another class of AMP architecture consist of non-monotonic AMPs where each core is power and performance optimized for a different instruction level behavior. AMP architectures (monotonic/non-monotonic) offer opportunities to achieve higher energy efficiency by dynamically migrating an application from one core type to another based on its current resource needs. The key challenges are: *(i)* how to dynamically determine which is the best core for the application to run on and, *(ii)* How often to allow such migrations, given that they may entail considerable overhead. Scheduling threads across cores is commonly done by the Operating system (OS). Popular OS-based schemes include Round Robin and FIFO. These scheduling schemes are suited for Symmetric Multicore Systems (SMPs), where each of the cores has the same capabilities. Thread-to-core mapping for non-monotonic (NM) architecture is more challenging as each core has diverse resources and exhibits varied performance/Watt. We have observed that different applications prefer different core types to achieve high performance/Watt. This motivates the need for an accurate dynamic thread-to-core assignment for NM architectures. Traditional OS-based schedulers are either static or respond to program changes at a large scheduling interval granularity while the behavior of programs may change much faster. Thus, a hardware based scheduling solution may perform better.

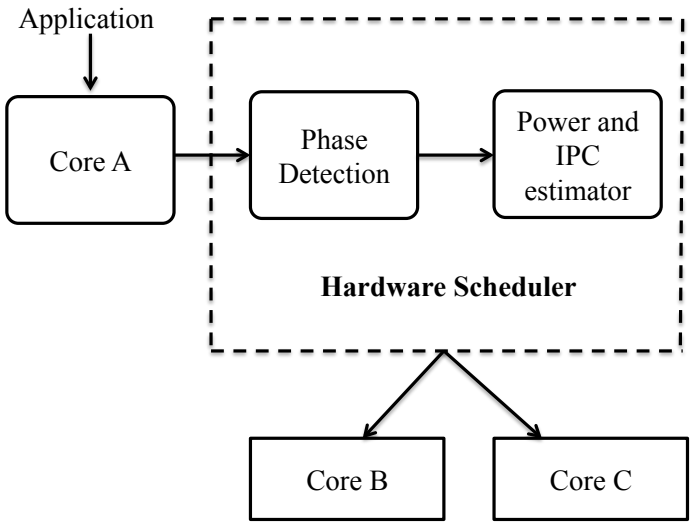

Figure 3.1. High-level overview of the hardware thread scheduling approach.

In this work, we develop a hardware based thread-to-core mapping which is *responsive* to program changes at a fine granularity and performs accurate thread-to-core assignment, resulting in higher performance/Watt. The central idea of this work is depicted in Figure 3.1, showing the hardware scheduler that re-assigns applications running on one core type to another NM core type that can achieve a higher performance/Watt. The hardware scheduler includes an online performance bottleneck based program phase classification mechanism, that detects program phases online, taking into account the core micro-architecture details. Each phase behavior can be characterized by a distinct resource bottleneck, which could be relieved by migrating the application to another core type that is provisioned with more resources to alleviate that bottleneck. To determine the most suitable thread-to-core mapping, we developed a scheme that estimates the expected throughput/Watt of the current phase in each of the different core types. The thread is then migrated to the core that is expected to provide the highest throughput/Watt. Thus, our scheme can closely track program phase changes and match the application phase requirements to the most suitable core type.

3.1 Non-Monotonic Architecture

We consider in this work non-monotonic AMPs that consist of diverse core types with varying core resource sizes, execution width, cache sizes and frequency. Prior works have performed core design space explorations to identify a set of non-monotonic core types that cater to diverse instruction level behaviors of different programs [69]. It was observed that, if only 1 core type is allowed, it would resemble current super-scalar OOO cores that strive to achieve a balance between core frequency and ILP. The remaining cores would be accelerator cores that are designed to relieve specific processor bottlenecks. A representative 4-core non-monotonic architecture that is considered in this work is shown in Table 3.1 [69]. We present in this thesis, an effective phase classification and a runtime scheme for thread-to-core mapping. The presented approach is not just suited to the particular flavor of non-monotonic architecture shown in Table 3.1. Instead, it can be applied to other diverse non-monotonic core architectures. Table 3.1 shows four core types. The baseline core type is called the Average core (AC) and it resembles an ordinary super-scalar core. In addition, we have three accelerator cores: 1) Narrow core (NC) that caters to application phases with ILP bottleneck, 2) Larger Window (LW) core that caters to application phases with an instruction window bottleneck, and 3) Wider core (WC) that caters to application phases with an execution width bottleneck.

In this work, we use a modified Gem5 simulator integrated with the McPAT power model [10, 59]. We use SPEC2006 and SPEC2000 benchmarks which are cross compiled for Alpha ISA with -O2 optimization [11, 89]. The benchmarks are run for 4 billion instruction after fast forwarding the first 2 billion.

3.1.1 Program Phase Detection

Our proposed thread-to-core mapping scheme detects program phase changes online and assigns the new program phase to the most suitable core type that can

Table 3.1. Core Parameters (A 2MB L2 cache is shared by all core types)

Core Type	F (GHz)	Buffer size (IQ,LSQ,ROB)	Width (fetch,issue)	Caches (KB) (I,D)
AC	1.6	32,128,128	3,3	64,64
NC	2	32,64,64	2,2	16,16
LW	1.4	48,128,384	4,4	128,128
WC	1.4	32,128,128	6,6	128,32

provide the best throughput/Watt. Applications have diverse program phases and as the computational needs of program phases are not available beforehand, they need to be determined online. Program phase behaviors have been studied extensively for the purpose of speeding up application simulation and for on the fly performance and power optimizations of different application phases [19]. Swapping of threads from one core type to another involves significant thread migration overhead. As a result, to reduce this overhead, thread swapping should to be done only when *stable* program phases are detected [46, 80, 86]. Therefore, phase detection schemes should detect stable program phases and leave out unstable or short duration phases that would not justify a reassignment to a new core type.

3.1.1.1 Phase Detection based on a Bottleneck Type Vector

Our proposed phase detection scheme is based on a bottleneck type vector (BTV) as the non-monotonic core types that are used in this work, cater to different performance bottlenecks. BTV tracks the frequency of the resource bottlenecks experienced by the application over an instruction interval. We track different performance bottleneck with the following performance monitoring counters [68]:

• Cache Stall: These counters track the I-cache, D-cache and L2-cache stalls. The I-cache stall counter tracks the number of cycles in which an instruction fetch is stalled due to an I-cache miss. The D-cache stall counter tracks the number of cycles in which a load is stalled due to a D-cache miss. The L2-cache stall counter tracks the number of cycles in which the processor stalled due to a L2 miss.

• Branch_Mispredict: This counter tracks the number of cycles the processor stalled due to a branch misprediction.

• Resource Stall: This counter tracks the number of cycles in which the instruction dispatch is stalled due to a blocked IQ or ROB or LSQ.

• Width Stall: This counter tracks the number of cycles in which ready instructions are stalled due to an insufficient issue width.

• IPC counter: This counter tracks the program's IPC during the period of execution interval.

The above counter values are normalized with respect to the total number of cycles during the instruction interval. In our scheme, each application first starts executing in the baseline AC core. After a fixed n committed instruction, the above mentioned bottleneck counters are read and a BTV vector consisting of the tracked counter values is formed and then, the phase classification is initiated. The phase classification's goal is to identify a previously classified stable BTV phase that has a similar bottleneck frequency distribution compared to the current BTV. When two BTV are compared, if the sum of the absolute differences between the components of the previously classified BTV and the current BTV is greater than a given phase threshold (that needs to be determined), then we classify it as a potential new phase. Otherwise, the current BTV is declared to match a previously classified phase. To differentiate between a stable BTV phase and an unstable phase, once a potential new phase is identified, we wait for m (another parameter that needs to be determined) consecutive intervals before classifying it as a new phase. Majority of m consecutive intervals must have BTV differences larger than the phase threshold (when compared to the previously classified stable phases). Once a new program phase is detected in the currently executing core, the best core type for the current phase is determined by computing the values of IPS^2/Watt for the current phase when executing in each of the different core types. A minimum improvement in IPS^2/Watt that would justify

a swap needs to be determined, since a too small a improvement would not outweigh the cost of thread swapping. Once the application is migrated to a new core type, our phase detection algorithm is re-initiated to detect new program phases and re-assign, if necessary, to the most suitable core type.

3.1.1.2 Phase Classification Parameters

To determine the quality of our phase classification approach, two metrics are used: 1) % of the total program execution that can be classified into stable phases, and 2) The ratio of the standard deviation in IPC between the classified stable phases to the mean value. To reduce the search space of the phase classification parameters, we considered only combinations of phase classification parameters that would result in % unstable phase and % standard deviation in IPC to be less than 15%. The search space for identifying the phase classification parameters resulted in instruction interval length varying between 10K to 150K, phase threshold parameter varying from 5% to 15% and phase interval (m) varying from 4 to 6.

As the goal of finding the right phase classification parameters is to maximize IPS/Watt, for each of the short-listed values of every parameter, we find the overall improvement in IPS/Watt using an oracular approach that maps the application phases to the NM cores listed in Table 3.1. As shown in Figure 3.2, the highest improvement in IPS/Watt is obtained for an interval length of 50K. In Figure 3.3 we analyze the effect of the phase threshold parameter, at an interval length of 50K, on other phase classification metrics. The metric %NonStablePhase provides the % of unstable phases obtained by our phase classification scheme, %SD-IPC is the IPC standard deviation for the classified stable phases, %Intervals-PhaseChange is the % of intervals that result in phase change, and ClassifedPhase is the average number of classified phases across all benchmarks. All of the above metrics are plotted for various phase thresholds (5% to 15%). A lower threshold (5%) results in a higher % of

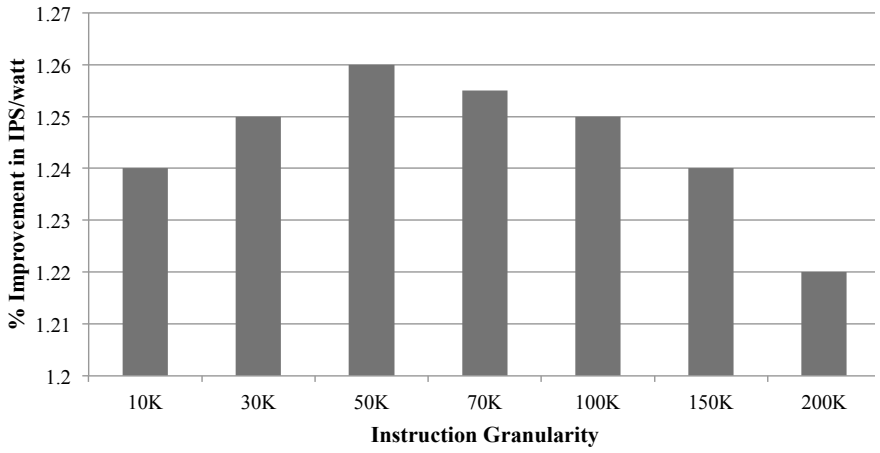

Figure 3.2. % of the improvement in IPS/Watt as a function of the interval length. Combinations of different phase classification parameters are averaged for the same interval length.

unstable phases, higher number of classified phases and an increase in the number of phase transitions. A higher threshold (15%) results in fewer classified phases and also an increase in the standard deviation of the IPC. Thus, considering the two phase classification metrics, we found that the largest throughput/Watt improvement is obtained for a phase interval length of 50K, phase threshold parameter of 8.5% and phase interval of m=4.

3.2 Online Phase to Core Mapping

We implemented a run time scheme using performance monitoring counters (PMC) for effective mapping of program phases to the non-monotonic core types. The decision to switch from one core type to another is based on the computed value of IPS^2/Watt for each of the different core types for the currently executing program phase. We develop a PMC-based estimation model for estimating the IPS^2/Watt of the currently executing program phase on each of the other core types using the PMCs of the currently executing core type. The development of PMC-based estimation model is similar to what is explained in Chapter 2.

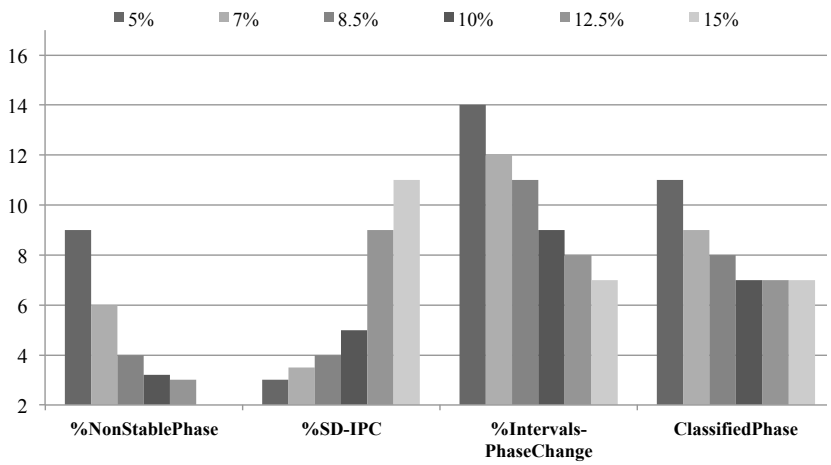

Figure 3.3. Various phase classification quality metrics for different values of the phase threshold parameter.

3.2.1 PMC-based Estimation Model

Prior publications have shown that using a small set of PMCs, power can be estimated online with high accuracy [78, 94]. Based on prior works, we use linear regression to derive expressions for estimating the performance and power for each of the different core type. For deriving these expressions, we chose a diverse set of workloads (*sjeng, h264ref, soplex, omnetpp, bzip2, namd, gobmk, hmmer*) from the SPEC06 suite [11] whose phases have affinity to at least one of the core types. To estimate the quality of the linear regression, we computed the correlation coefficient (R^2). As shown in Table II, the average correlation coefficients for power and performance were found to be 0.91 and 0.84, respectively. We also show in Figure 3.4, the average error in computing power and performance across different core types using the PMCs of the current core type. The average errors when estimating the power and performance are 11% and 8.5%, respectively. Thus, we are able to estimate power and performance with reasonably high accuracy allowing us to make the right thread-to-core mapping. The derived linear regression expressions are used by the hardware scheduler to compute the power and IPC online as explained next.

Table 3.2. Power and Performance estimation accuracy across different core types

Core Type	Correlation Coefficient (Power)	Correlation Coefficient (IPC)
PMC AC ⇒ Power/IPC	0.93	0.83
PMC NC ⇒ Power/IPC	0.88	0.81
PMC LW ⇒ Power/IPC	0.91	0.85
PMC WC ⇒ Power/IPC	0.92	0.87

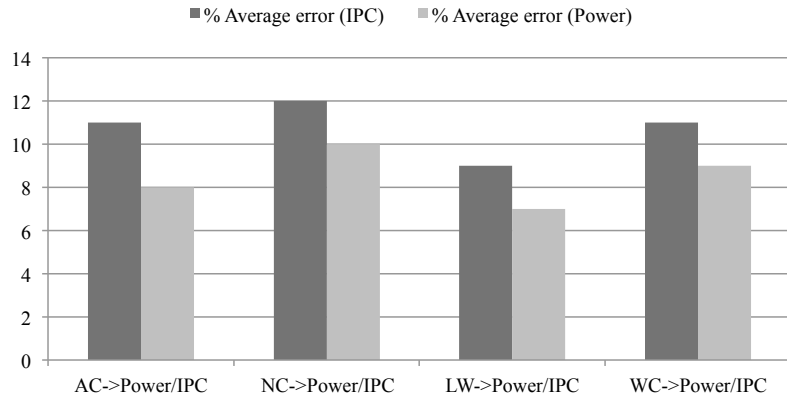

Figure 3.4. % of the average error in computing IPC and power for each of the different core types. AC ⇒ Power/IPC denotes the average error in estimating power and IPC in each of different core types using the PMCs of the AC core.

3.3 Thread Migration Overhead

Our thread migration decisions are made online and are transparent to the operating system (OS), that is unaware of the underlining heterogeneity of the cores. In our scheme, the initial thread-to-core mapping is done by the OS. Then, our proposed mechanism will attempt to increase the throughput/Watt by choosing the right core type for each detected phase. Our scheme relies on phase detection logic and hardware controller to compute the expected throughput/Watt in each of the different core types. The goal of the phase detection logic is to detect stable program phases using predetermined phase classification parameters. Once a stable program phase is identified, we compute the expected throughput/Watt from the monitored PMC values using a MAC (multiply and accumulate) unit. The MAC unit uses the linear regression expressions (that are functions of PMCs values) to compute the power and performance. The MAC unit is pipelined and can compute 1 MAC operation per cycle. We estimate the overhead for all the MAC computations to be 60 cycles. Each time we decide to migrate the application phase to another core type, we incur an overhead for transferring the register state and for cache warm up in the new core. The overhead for the register transfer and the cache warm up is conservatively assumed to be 350 cycles [69, 80]. We assume that the time required to wakeup the core, into which the thread is migrating, is hidden since when this core is woken up, the previous core is still executing the program. Thus, taking all the overheads into account, we conservatively assume a total overhead of 2000 cycles upon a thread migration. In the results section, we also show the impact of a higher overhead on the overall performance.

3.4 Results

3.4.1 Evaluation Framework

In this section we evaluate different program phase to core mapping schemes and compare the throughput/Watt achieved by each of the schemes to our BTV-based phase detection and PMC-based thread-to-core mapping approach. The compared to schemes include:

• Static Scheme: This is the baseline scheme against which our various thread-to-core mapping schemes are compared. In this scheme we execute the entire application on the AC core type, as it resembles a conventional super-scalar core.

• Oracular Scheme: Here, an oracle determines the phase to core mapping once a phase change is detected, by making accurate predictions of the expected throughput/Watt. Oracular based approaches are used to obtain an upper-bound for the achievable throughput/Watt and can not be implemented in practice. Still, to have a realistic scheme, we take into account the overhead for a core switch.

• Sampling Scheme: In this scheme, once a phase change is detected, the new program phase is sampled on each of the core types for a period of 10K instructions. During this sampling period, power and IPC values are collected in each of the core types and the core that provides the highest IPS^2/Watt is chosen as the right core type for the application phase. We continue executing on the new core type until we detect the next phase change. We take into account the overhead of sampling in computing the throughput/Watt of this scheme.

• ITV-based Scheme: Several recent publications have used a thread-to-core mapping scheme for AMPs that relies on an Instruction Type Vector (ITV) to detect program phase changes [46, 80, 81]. The AMP core types considered in these works are monotonic cores types consisting of only high performance or low performance cores. We have implemented the ITV-based phase detection scheme in order to investigate how well will this scheme perform when applied to non-monotonic core types.

If what follows we outline the implementation of the ITV scheme. After the commit of n (a parameter to be determined) instructions, we collect the values of counters that count the number of different retired instruction types out of the n instructions. The collected instruction types include load, store, integer, floating-point and branch. The counters that form the ITV are different from those that form the BTV, as the BTV counters are micro-architecture dependent while the ITV counters only count the different types of retired instructions. After every n retired instructions, the new ITV vector is compared to ITV vectors of previously classified phases. If the resulting sum of the absolute differences between the ITV components is greater than a predetermined threshold (called phase threshold), then the newly captured ITV vector belongs to a new phase. If the calculated value is smaller than the threshold, the previously classified phase is repeating. As the goal of the phase classifying algorithm is to identify stable program phases, we wait until m consecutive intervals have an ITV vector difference that is smaller than phase threshold. As the benchmarks that we use are somewhat different from those used in [46, 80, 81], we have redone the phase classifying experiments based on the ITV, and have determined the following values of the phase classification parameters: interval length n=150K, m=4 and phase threshold=11.2%. The throughput/Watt achieved by the ITV scheme is calculated for the sampling, oracular and PMC-based schemes. The PMC-based scheme refers to using PMCs to estimate the IPS^2/Watt for each of the core types before the decision to migrate the thread is done.

3.4.2 Throughput/Watt Analysis

We analyzed the throughput/Watt obtained by our BTV approach and then compared it to other thread-to-core mapping schemes as mentioned above. Figure 3.5 shows that we obtain a 22% average improvement in throughput/Watt for the SPEC benchmarks [11, 89], compared to the static scheme where the application is run to

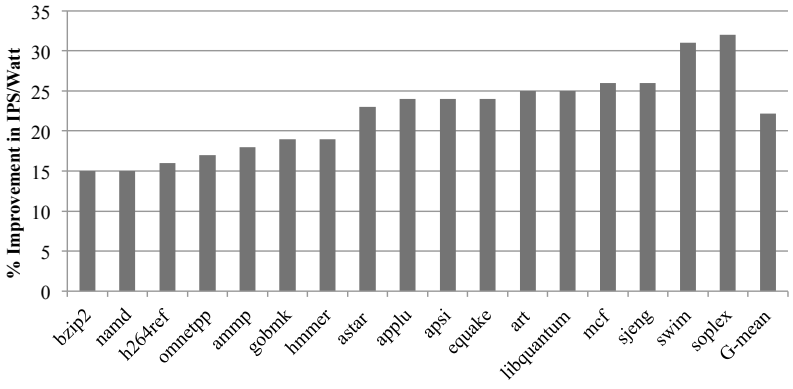

Figure 3.5. % Improvement in throughput/Watt obtained by the PMC-based BTV scheme when compared to the static scheme.

completion on only one core type (AC). Diverse core bottleneck behaviors observed in each of the benchmarks are captured by the application phases and are mapped to core types that relieve these bottlenecks, thus improving the throughput/Watt. The considered bottlenecks include L1 cache (Icache, Dcache) bottleneck, L2 bottleneck, instruction window bottleneck, branch misprediction bottleneck and instruction issue bottleneck. The benchmarks that we experimented with have mixed characteristics with diverse phase behavior consisting of compute intensive benchmarks (*hmmer, bzip2, h264ref*), branch intensive benchmark (*astar, gobmk, art*) and memory intensive benchmark (*mcf, libquantum, soplex*). The average and maximum number of stable phase detected by the BTV approach across all the benchmarks have been 8 and 11, respectively. Once a stable program phase is detected, we can either map the program phase to another core type or continue execution on the same core type. On average we have observed only 8.2% transitions from one core type to another and the number of times the hardware scheduler is woken up to collect PMC values and compute the IPS^2/Watt is on the average 650 times per application run. The number of thread migrations and the invoke rate of the hardware scheduler are not too high resulting in a low core switching overhead.

Figure 3.6 compares the average throughput/Watt for the BTV scheme using the oracular, sampling and PMC-based thread-to-core mapping approaches. The Orac-

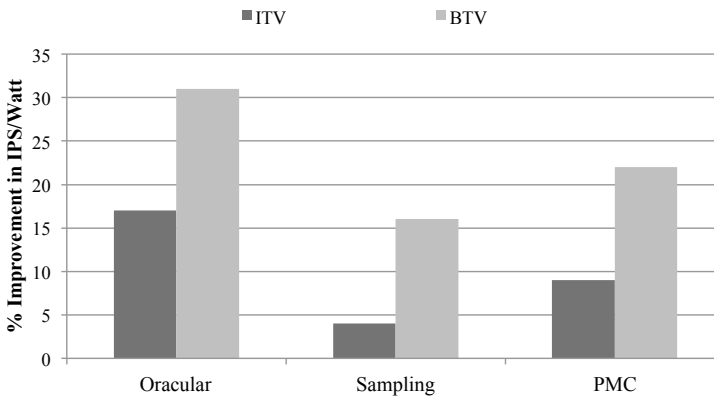

Figure 3.6. % of improvement in throughput/Watt achieved by various switching schemes for the BTV and ITV based approaches.

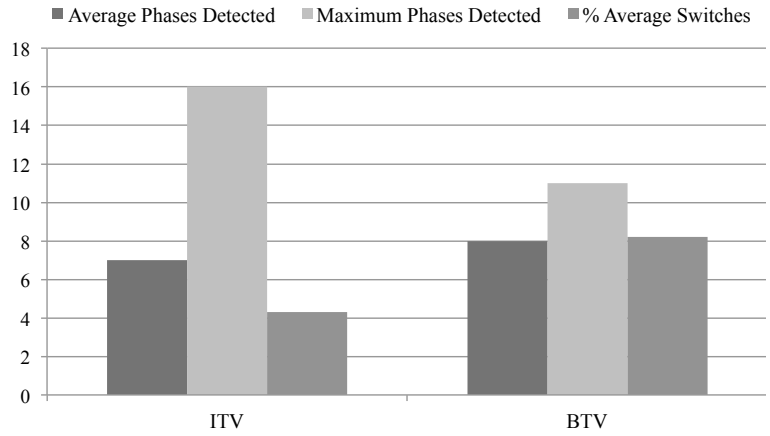

Figure 3.7. Comparing the number of phases detected and the number of switches for the ITV and BTV based schemes.

ular scheme, on average, obtains a 10% higher improvement in throughput/Watt compared to the PMC-based scheme. The PMC-based scheme outperforms the sampling scheme by 8% due to the higher overhead experienced by the sampling. The BTV scheme shows an improvement of 12% in throughput/Watt compared to the ITV-based approach as it considers core bottlenecks rather than only the distribution of instruction types. The BTV scheme switches between core types more frequently than the ITV scheme, spending more time in affine cores that alleviate performance bottlenecks. Figure 3.7 shows the number of phases detected by both schemes. On Average, the ITV scheme detects 7 program phase vs. 8 detected by the

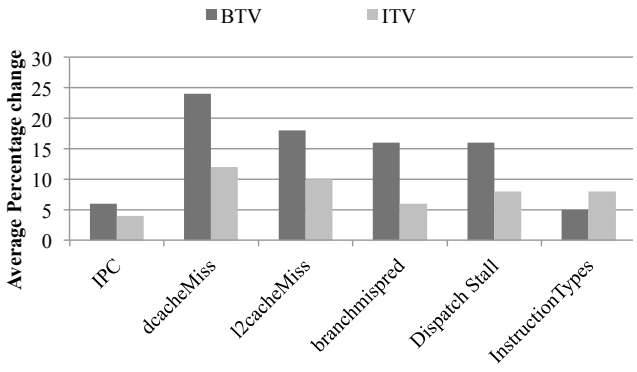

Figure 3.8. Average % change in PMC events between different program phases classified by the BTV and ITV schemes.

BTV scheme. We also analyzed the number of phase to core transitions performed by the two schemes. The BTV-based scheme has more transitions than the ITV-based one. Out of the 4.3% transitions for the ITV scheme, 2% agree with the BTV scheme. To further compare the phases identified by the BTV and ITV schemes, we studied the relative change in PMC values between different program phases classified by the ITV and BTV schemes. Figure 3.8 shows the percentage change in *IPC, dcacheMiss, l2cacheMiss, branchMispred* (number of branch mis-predictions), *DispatchStall* and Instruction types between different program phases classified by the ITV and BTV approaches. The BTV scheme captures bottlenecks in program execution and thus shows a higher percentage of variation in the bottleneck counters between different program phases. In contrast, ITV phases are micro-architecture independent and only capture differences in instruction types and thus shows a higher percentage of variation in instruction types between phases.

Figure 3.9 compares the improvement in performance obtained by the ITV and BTV approaches using various switching schemes. We obtain a higher, by 6.5%, improvement in performance using the BTV approach compared to the ITV one.

Figure 3.10 shows the impact of the core hopping overhead on performance. As mentioned previously, the average overhead of core switching has been assumed to

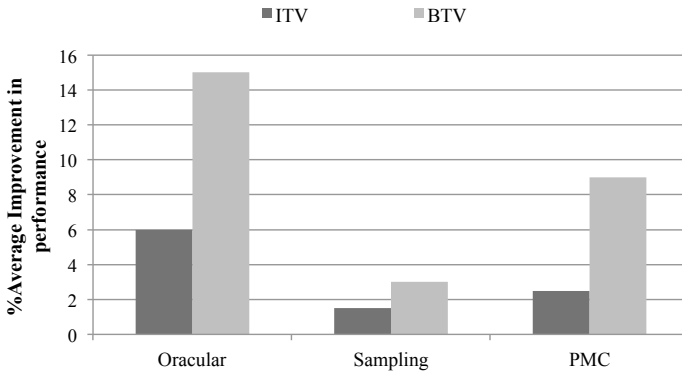

Figure 3.9. % of improvement in performance obtained from various switching schemes for the BTV and ITV based approaches.

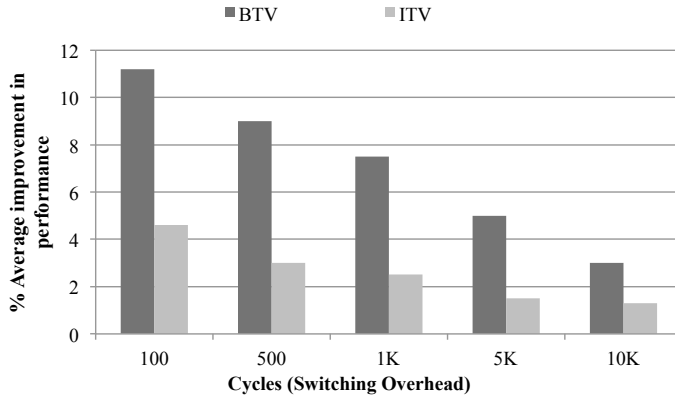

Figure 3.10. The impact of varying switching overhead (in cycles) on performance.

be 500 cycles. If the overhead for switching increases from 500 to 1K cycles, the performance drops by only 1.5% for the BTV scheme. Increases beyond 1K cycles, result in a steeper drop in performance. In summary, our results indicate that the BTV-based phase detection scheme outperforms the ITV-based scheme, achieving a higher throughput/Watt for non-monotonic core types.

As prior ITV approaches have been used for mapping application phase to monotonic core types [46, 80, 81], we also compare BTV vs. ITV based approaches for monotonic cores. We studied monotonic architecture consisting of Big/Little cores. Big core is OOO core that resembles the AC core type. Little (InO) core is implemented with fetch/issue width of 2, IQ entries of 36, similar cache sizes and frequency to that of AC core. Both the cores share a common L2 cache. We implemented both ITV and BTV based phase detection approaches for the monotonic core design. As before, the IPS^2/Watt is estimated from the PMCs. Program phase is then mapped to the core that provides the best throughput/Watt. Switching from OOO to InO core is preferable during memory bound, high branch mis-predict or low ILP phases for better energy efficiency. Micro-architecture independent counters such as those used for tracking ITV phases can track whether the phase is memory intensive (counting load/store access and using IPC information) or compute intensive (number of integer/floating point instruction executing in pipeline and IPC information). Unlike non-monotonic core types which have varied performance and power profiles across different core types, monotonic core types provide high performance or low power core types, which helps ITV to map compute intensive phase to high performance core and memory intensive phases/low IPC phases to low power cores. As a result, the advantages of BTV in monotonic cores is not as pronounced.

Figure 3.11 compares the throughput/Watt improvement obtained for our monotonic core design. Using BTV based approach, we obtain 14% improvement in throughput/Watt and 11% from the ITV approach using PMC based scheme. The

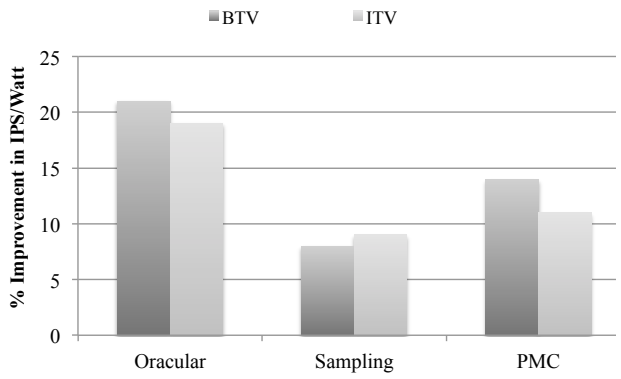

Figure 3.11. % Improvement in throughput/Watt obtained from various switching schemes for monotonic core types for our BTV and ITV based approaches.

ITV based approach does considerably well for monotonic core types, as the difference in throughput/Watt between the two approaches (ITV and BTV) is only 3%. The BTV based approach for non-monotonic core type provides a higher by 9% throughput/Watt when compared to monotonic core types.

3.5 Conclusion

To achieve the high throughput/Watt potential of non-monotonic cores, a runtime mechanism is needed for thread-to-core assignment that would choose the most suitable target core. In this work we propose an efficient thread-to-core mapping scheme that detects program phase changes based on various resource bottlenecks and uses performance monitoring counters for assessing the most suitable target core. Our results indicate that the proposed thread-to-core mapping scheme can increase the throughput/Watt of application by 22% for non-monotonic architectures and by 14% for monotonic architectures. The methodology described here is equally applicable to other variants of monotonic and non-monotonic core types.

CHAPTER 4

IMPROVING POWER EFFICIENCY USING MORPHABLE ARCHITECTURE WITH NON-MONOTONIC CORE TYPES

Previously explored AMP cores have shown that applications can exhibit diverse program phase behavior where each program phase behavior can exhibit one (or more) of common processor performance bottlenecks arising from cache misses, limited execution resources or execution width, large degree of instruction dependencies, or inherently low instruction level parallelism [69]. Consequently, there is need for diverse set of non-monotonic AMP architectures where each core is power and performance optimized to different instruction level behavior designed to address these bottlenecks. In Chapter 2, we explored reconfigurable monotonic AMP architectures that provides higher energy efficiency than the baseline OOO core and Big/Little architectures. In this chapter, we explore reconfigurable non-monotonic architectures that can provide improved power efficiency compared to monotonic designs presented in Chapter 2.

Dynamic morphing of core resources in monotonic cores was previously proposed [47, 61, 91, 92]. Such designs eliminate the overhead associated with transferring the state of a workload from one core to another upon a switch. This allows morphing to take place at fine grain instruction granularities ($\sim$1000 instructions) which reportedly results in significant energy savings at a small loss in performance [61] . We observe that resource bottlenecks or excesses can be quite diverse in applications. Thus, morphing between just two core modes may not fully exploit power and performance improvement opportunities.

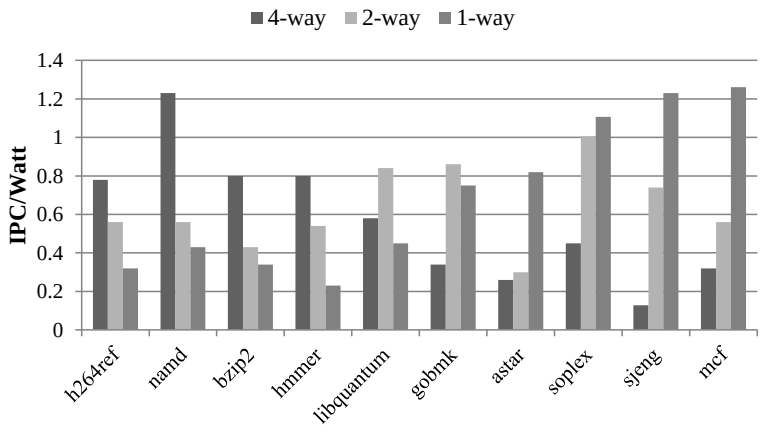

Figure 4.1. IPC/Watt for SPEC benchmarks [11] running on OOO cores differing in fetch/execution/retire widths and core resources.

To test the potential benefits of having three or more distinct morphable core configurations, we analyzed the IPC/Watt of three OOO cores differing in execution widths and core resources (which are scaled appropriately for the chosen width), for workloads from the SPEC 2006 suite [11] . We call these cores the 4-way, 2-way and 1-way cores where 4,2, and 1 indicate the execution width. Figure 4.1 shows that there are workloads that achieve the highest IPC/Watt when run on the 4-way core while there are other workloads for which a 2-way or even a 1-way core can provide the highest performance/Watt. The latter workloads do not need the same amount of resources as those that prefer the 4-way core. Hence, for such workloads energy savings can be achieved by running them on the reduced fetch width core with reduced resource sizes, resulting in better performance/Watt.

Figure 4.2 shows the performance/Watt for the *sjeng* benchmark from SPEC 2006 suite [11] at fine instruction granularity, for the three OOO cores mentioned above. The temporal variation in demand for resources exhibited by the workload, motivates studying a morphable architecture that can morph between three (or more) core modes as it may achieve considerable performance/power improvements.

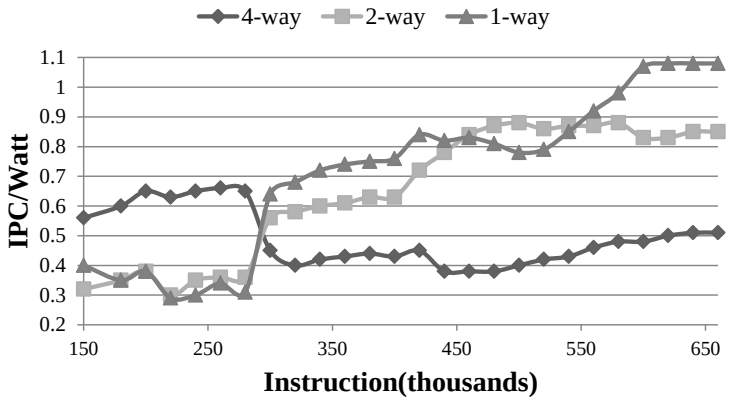

Figure 4.2. IPC/Watt over a period of execution for the benchmark *sjeng* [11] sampled every few thousand of committed instructions.

We performed a core design space exploration to select a set of core architectures that are fundamentally different from big/little architecture. The number of modes is determined based on the law of diminishing marginal utility. The design space exploration has resulted in four distinct core modes appropriate for fine grain switching. The architectures of the core modes differ in fetch width, issue width, buffer sizes (e.g., ROB, LSQ and IQ), clock frequency and operating voltage. We then use the selected core architectures to define the distinct core modes of the proposed morphable architecture. Our self-morphable core will dynamically morph into any one of these execution modes differing in fetch width, issue width, buffer sizes (e.g., IQ, LSQ, ROB) and clock frequency. This way, the self-morphable core can mimic a high diversity asymmetric multicore processor. A morphable core architecture that can switch between the different core modes needs an effective online mechanism to determine the most efficient core mode for the current phase of application. Furthermore, since many performance and power improvement opportunities exist at a fine instruction granularity, both the (i) morphing decision mechanism, and the (ii) hardware morphing process need to be fast. To this end, we propose an online decision

mechanism to identify the most power efficient core mode for the current execution based on hardware performance monitoring counters (PMCs).

4.1 Related Work

4.1.1 Non-montonic Multicore Processors (AMP)

Kumar *et al.* proposed non-monotonic core types containing mix of cores with different power and performance characteristics, so that every application phase is scheduled to a core that achieves the best power-efficiency [52]. They showed significance performance benefits for multi-programmed workloads. Their solution does not take advantage of pipeline depth or varying frequency in the core design search space and are focused on maximizing throughput only. They only considering oracle steering towards different core type. Navada *et al.* considered accelerating single threaded workloads by performing complete design space exploration and identifying a set of heterogeneous cores that would maximize performance [69]. Their core design space exploration was done at coarse grain instruction granularity to determine the best set of N-core type for AMP design. Their conclusion was that with N core types, the optimal number of heterogeneous cores for single threaded performance would contain an average core (i.e., best homogeneous core) and (N-1) accelerator core types that target specific bottlenecks encountered during a program execution. Azizi *et al.* explored energy-performance trade-offs in processor design space [6]. They examined six different processor architectures consisting of single-issue, dual-issue and quad-issue designs with both in-order and out-of-order execution and proposed different architectures for different design objectives. In this work we try to unearth non-monotonic core types when switching could be performed at fine grain granularity through a design space exploration experiment.

4.1.2 Adaptive Asymmetric Cores

There are several publications advocating morphing of a core at runtime to adapt to changing workload needs and improve performance and/or power efficiency. The closest work to our proposal that advocated morphable asymmetric cores for switching at fine grain instruction granularity consisted of monotonic core types [47, 61, 71, 91, 92]. The above mentioned morphable architectures focus only on morphing between two extreme architectures while we explore, in this work, morphing into a larger number of core configurations (or modes). Such a morphable architecture is more likely to match the demands of various workloads by addressing a more *diverse set of bottlenecks*.

A less aggressive form of core morphing has been discussed in [2, 7, 8, 26, 74]. These configurable architectures dynamically adjust the cache and storage buffers such as ROB, LSQ and IQ to the application demands. The proposed configurable architectures do not consider varying the execution width or changing the frequency and voltage. Dubach [27] *et al.* proposed machine learning based predictive model that predicts the best hardware configuration for any phase and then dynamically change the micro-architecture. Kumar *et al.* explain how expensive structures between adjacent cores can be shared while keeping floorplan in mind [53]. Homayoun *et al.* proposed ways by which micro-architectural structures can be shared across 3D stacked cores [41]. The above techniques are limited to a number of micro-architecture structures they adapt to and are suitable only for coarse grain switching. Other approaches have proposed sharing caches and pipelines between cores. Cache sharing approaches have focused on multiplexing L1 caches among multiple cores to eliminate state transfer overhead on thread switch from one core to another [25, 65, 77]. Multiplexing pipelines within cores have also been proposed to achieve heterogeneity [67].

4.1.3 Dynamic Voltage and Frequency Scaling

A heterogeneity technique that has been widely used for obtaining energy efficiency is Dynamic Voltage and Frequency Scaling (DVFS). DVFS reduces voltage and frequency of the cores to obtain high energy efficiency at the expense of performance loss. DVFS is usually targeted in memory intensive phases when maximum energy efficiency could be obtained with minimum performance loss. Traditionally, the benefits of DVFS techniques are limited by the voltage and frequency scaling overhead. Whenever there is a change in processor frequency, the PLL needs to relock to the newer frequency which would result in halting of processor. This overhead in Intel processors in taken to be 5 μs [72] . There is an additional overhead incurred when scaling up voltage/frequency as the processor operates at lower frequency till the voltage has scaled up to the newer value, resulting in performance loss [72]. This overhead is estimated to be around 25 μs for our range of voltage/frequency considered based on the work in [72]. Thus, to minimize the high overheads, DVFS is applied at coarse grain instruction granularity on the order of millions of processor cycles and is therefore, limited to operating system scheduling intervals. Previous studies have also explored the balance between performance/energy for DVFS with several frequency/voltage levels [43, 104]. To mitigate the high overhead of coarse grain DVFS schemes, researchers have proposed using on-chip regulator that can allow rapid transitions between the voltage levels [29, 49, 56]. With the help of on-chip regulator, voltage transition time is brought down to less than 20ns. Our proposed non-monotonic core types with varied frequency/voltage can improve energy efficiency not only in memory intensive phases but also in branch intensive phases, low ILP phases and phases that have serial computation. Recent research has shown that heterogeneity within a core can provide a higher energy efficiency than DVFS schemes [36, 60]. Energy saving benefits of DVFS are also diminishing with newer technologies as observed in [57].

4.2 Proposed Architecture

Asymmetric multicores may contain multiple core types with each core type specialized for a specific workload characteristics. As the multicore is constrained by the Thermal Dissipation Power (TDP) limit of the package, the cores cannot feature the largest possible size for all micro-architectural structures and yet operate at the highest possible frequency.

Consequently, there are always trade-offs in core design. For example, to support a higher degree of instruction level parallelism (ILP), the pipeline width should be increased. Such an increase would, in turn, limit the allowed core frequency due to the TDP constraint. Thus, a core specialized for high ILP may not meet the needs of a workload with extensive sequential instruction dependency whose performance can only be improved by increasing the frequency. Therefore, designing the right mix of cores for an AMP that caters to the demands of diverse workloads requires careful balancing.

A diverse mix of cores that can address different resource demands can also benefit the execution of a single benchmark. For example, Figure 4.3 shows the IPC variations between 0.6 to 1.6 observed in the course of executing the SPEC benchmark *sjeng*. The figure shows that as the IPC varies, the usage of several core resources varies too. We further observe that variations in ROB, LSQ and IQ occupancy happen at a *small instruction granularity*.

Therefore, our proposed *self-morphable core* should be capable of dynamically reconfiguring to adapt to the current demands of the executing workload and should allow such reconfigurations to be done at a fine-grain instruction granularity. To this end, we do not vary cache sizes upon core reconfiguration to avoid costly migrations of cache content. The selected fixed cache sizes were determined experimentally to be: 64KB for the I-cache and the D-cache and 2MB for the L2 cache.

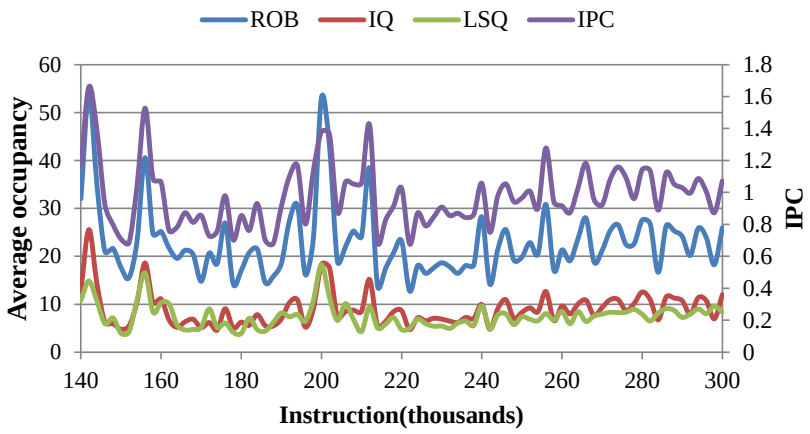

Figure 4.3. IPC and resource occupancy over a period of benchmark *sjeng*'s execution as a function of instructions committed.

Table 4.1. Core Design Parameters

Core Parameter	Range of Values
Fetch, Issue Width	1,2,3,4,5
ROB size	8,16,32,64,96,128,192,256,384
Issue Queue size	12,24,36,48,64
LSQ Size	8,16,32,64,96,128,192
Clock Period	0.4ns-1ns (steps of 0.1ns)

4.2.1 Design Space Exploration

We wish to identify the distinct core modes that should be supported by our morphable core. Table 4.1 outlines the range of core parameters for our core selection. Clock frequency is varied within the common range of super-scalars' speed. The search would determine the core modes that can provide the best performance/Watt for fine grain instruction slices. If we allow the relevant core parameters to assume all their possible values shown in Table 4.1, the resulting design space exploration would require experimenting with 11,025 combinations of parameters. However, core sizing for improved performance/power in [46] has shown that increasing the size of one resource without a commensurate increase in other resources yields limited benefits. Thus, certain parameter combinations such as (ROB=8, IQ=64, LSQ=192, Width=4) are not acceptable design candidates as a small 64-entry ROB cannot support large IQ and LSQ and would not result in any performance or power benefits. By performing design space pruning, we have reduced the number of core design combinations that need to be analyzed to 300.

The remaining 300 design combinations were analyzed exhaustively with the objective of achieving the highest possible IPS^2/Watt by allowing switching between core modes every 2000 instructions. The decision to switch modes is based on the metric IPS^2/Watt that assigns higher significance to performance than to power. The reason for choosing 2K as our fine grain instruction interval will be explained in the next section. After each 2K retired instructions interval we compute the potential increase in the IPS^2/Watt for every core configuration out of the 300 candidates. A minor increase in the IPS^2/Watt does not justify a core mode switch but, more importantly, does not justify adding a new core mode. Therefore, we had to choose a threshold for the minimum improvement that will justify an additional core mode. Each mode switch involves an overhead (explained later) and thus having large number of core modes may result in high morphing overhead while too few modes may

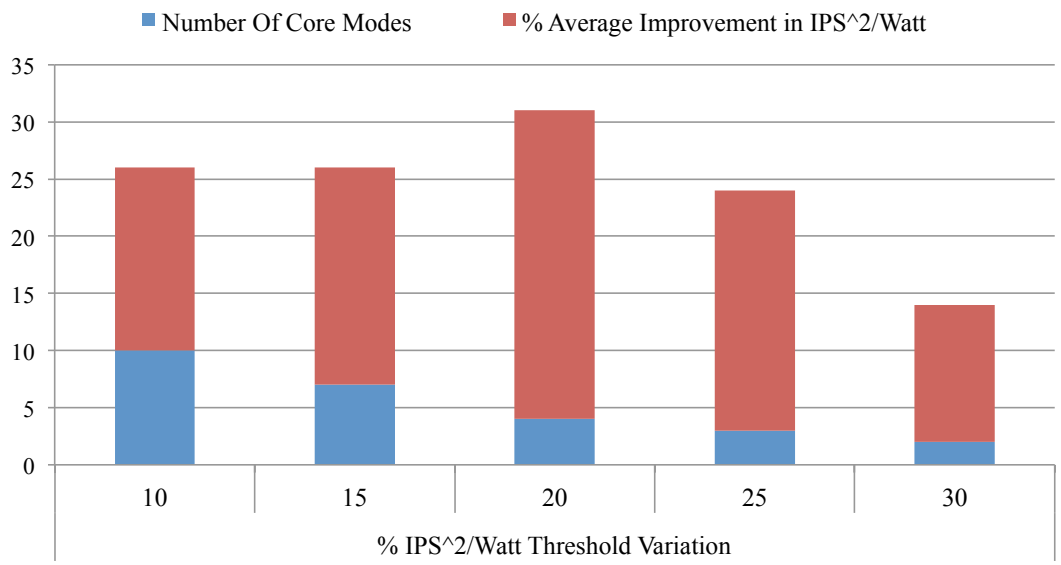

Figure 4.4. % Average improvement in IPS^2/Watt and number of modes as a function of IPS^2/Watt threshold.

not benefit all benchmarks and thus achieve a lower IPS^2/Watt. Our experiments as shown in Figure 4.4, revealed that selecting a IPS^2/Watt threshold of less than 20% yields more core types but the additional IPS^2/Watt improvement achieved by most of these core types is limited. Increasing the threshold to 20% reduces the number of core types to four with a higher IPS^2/Watt improvement for most benchmarks. We have also observed that the IPS^2/Watt improvement and the details of the core types are not very sensitive to small variations (from 20%) in the threshold. A further increase in the threshold to 25% (and higher) resulted in fewer core combinations but the majority of benchmarks did not benefit from morphing. We have, therefore, decided to use a threshold of 20% and have as a result, four core modes.

4.2.1.1 Power Unconstrained Core Selection

The architectural parameters of the selected four core modes that will best accommodate the diverse application phase behavior (of the SPEC benchmarks) are shown in Table 4.2. This set includes an Average Core (AC) which targets most of

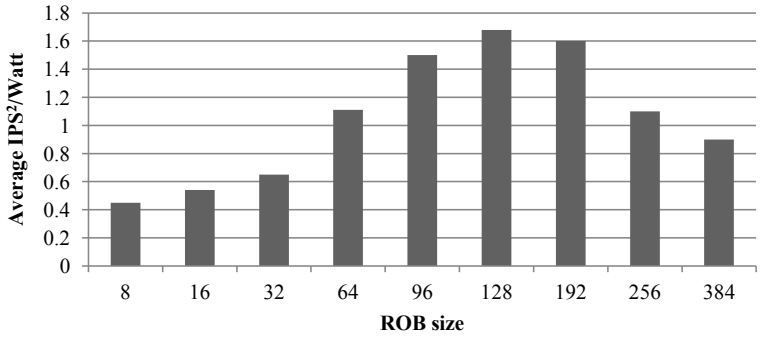

Figure 4.5. IPS^2/Watt as a function of ROB size for the AC core mode (power unconstrained).

Table 4.2. Core parameters for a power unconstrained design

Core Mode	F (GHz) /V	Buffer size (IQ,LSQ,ROB)	Width (fetch,issue)	Average Power (W)
AC	1.6/0.8	36,128,128	4,4	2.2
NC	2/1	24,64,64	2,2	1.7
LW	1.4/0.8	48,128,256	4,4	2.4
SM	1.2/0.7	12,16,16	1,1	0.82

the application phases and a Larger Window (LW) core that has a bigger window size and targets application phases which have a window bottleneck. In addition, the set includes a Narrow Core (NC) which targets application phases with low ILP and accelerates sequential execution using a higher frequency. The fourth mode is a Small core (SM) with a lower frequency that caters to low performance phases that exist at fine grain granularity.

To illustrate our core mode selection process, we show the impact of changes in the ROB size of the AC mode in Figure 4.5 for a subset of benchmarks that spent significant amount of time in the AC mode. It can be observed that ROB=128 (the chosen size for the AC mode), offers the best IPS^2/Watt. Figure 4.6 illustrates our process for determining the frequency. We observe that at a frequency of 1.6GHz the IPS^2/Watt is the highest.

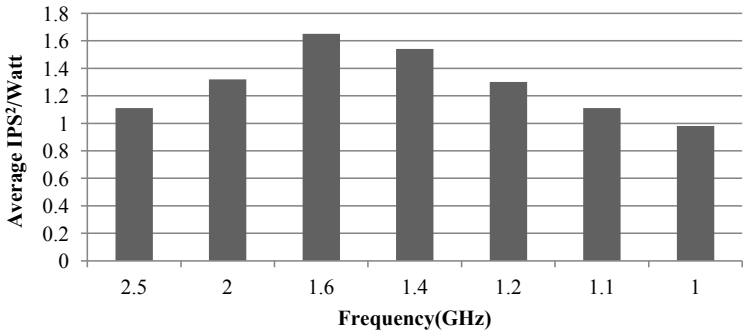

Figure 4.6. IPS^2/Watt as a function of frequency for the AC core mode (power unconstrained).

Table 4.3. Core parameters for a power constrained design (2W)

Core Mode	Freq (GHz)	Buffer size (IQ,LSQ,ROB)	Width (fetch,issue)	Average Power (W)
AC_2W	1.6	36,128,96	4,4	1.6
NC_2W	2	24,64,64	2,2	1.7
LW_2W	1.2	48,192,128	3,3	1.9
SM_2W	1.2	12,16,16	1,1	0.82

4.2.1.2 Power Constrained Core Selection

In the previous section we searched for the best core modes without restricting the overall peak power. Peak power dissipation is important for processor design since the thermal budget of processor, cooling cost, power supply cost and packaging cost depend on the processor's peak power dissipation [50]. We now repeat the search (for the preferred core modes) but with a limit on the power budget. We considered processor peak power dissipation limits similar to those in prior works [69, 99], i.e., 2W and 1.5W. For a peak power constraint of 2W, we obtained four somewhat different core modes shown in Table 4.3. Further reducing the power budget to 1.5W, reduces the number of preferred core modes to three with the high-frequency narrow core excluded.

Table 4.4. Core parameters for a power constrained design (1.5W)

Core Mode	Freq (GHz)	Buffer size (IQ,LSQ,ROB)	Width (fetch,issue)	Average Power (W)
AC_1.5W	1.4	36,64,64	3,3	1.32
LW_1.5W	1	24,128,128	3,3	1.4
SM_1.5W	1.2	12,16,16	1,1	0.82

4.2.2 Dynamic Morphing

In the proposed scheme all core modes are derived from a single OOO processor core with banked resources, where each bank can be turned on or off and the frequency can be raised or lowered to configure the core to the modes described in Table 4.2. The buffers that are dynamically resized are the ROB, LSQ and IQ. The fetch width and issue width are also dynamically resized. Decoding units are subsequently powered on/off when the fetch and issue width is resized.

Our baseline execution mode is an average OOO core (AC) that will be dynamically morphed into three other modes, namely, smaller core (SM), narrow core (NC) or larger window (LW) core during runtime. Although each of the four modes has a distinct combination of buffer sizes, fetch and issue width and frequency, they all have the same cache size. This allows us to resize resources while leaving the contents of the cache intact, which in turn allows fine grain switching with low overhead to take advantage of every opportunity for power savings or performance enhancement.

Switching from one mode to another is determined by estimating the power and performance in all other modes based on performance counters' values in the currently executing mode. We reconfigure into another core mode only when the reconfiguration is predicted to result in a sufficiently higher IPS^2/Watt. An in-depth description of our runtime switching mechanism is provided in the next section.

4.2.3 Adaptively Sizing Buffers for the Morphable Core

In the proposed morphable core, the ROB, IQ and LSQ are implemented as banked structures where each bank can be independently powered on/off. The banks have

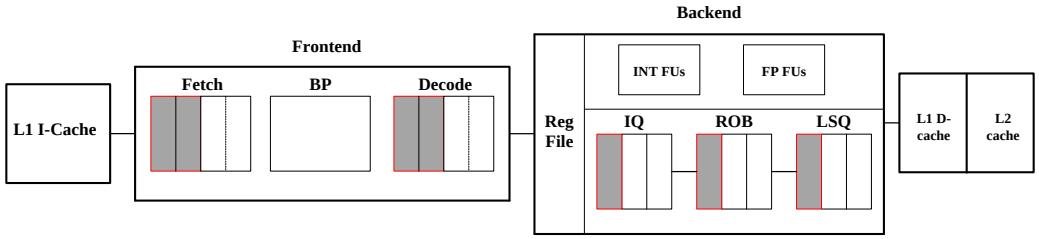

Figure 4.7. High-level view of the morphable core. The shaded units are reconfigured during run time.

their own set of input/output drivers, pre-chargers and sense amps. The dynamically re-sizable buffer can be formed by stacking more than one of these banks together [2, 18, 74] . The bank size for ROB, LSQ and IQ needs to be determined carefully. A too small a bank may result in larger resizing overhead in terms of layout area and design cost. It has also been shown that a too big bank size causes a significant increase in energy consumption whereas bank sizes of 8, 16 or 32 have only small differences in energy consumption [101]. Thus, taking technology considerations into account, the bank size for the ROB and the LSQ is set to 16 and for the IQ it is set to 8 [74, 101] .

4.3 Runtime Morphing Management

The proposed dynamic morphing/reconfiguration relies on online estimators to select the best core mode for the current needs of the executing application. The previously proposed morphing in [61] has determined the core to morph into by computing the performance (online) without taking power into account. Also, prior configurable architectures did not provide an effective online management scheme for run time morphing decisions [26, 74]. We designed an online power and performance estimation scheme that is fast and sufficiently accurate to support the morphing decision process. The key challenge here is the fact that while the program is executing on the current core mode, we need to estimate the IPS^2/Watt for all four core modes.

To estimate power and performance on-line for computing the IPS^2/Watt metric we need to select an appropriate set of hardware performance counters. We start with a large number of counters that have good correlation with power and identify a smaller set of counters that can be used to estimate power and performance online in each of the core modes at a sufficient accuracy. Linear regression is then used to derive expressions for estimating the performance and power in the other core modes using hardware performance monitoring counters (PMCs) in the current core mode. Prior works [23, 80, 93] that derived expressions for estimating power and/or performance have considered only a big/little architecture and without changes in voltage/frequency. In this work we show how accurately we could predict power and performance in each of the cores modes which are architecturally different and are running at different voltages and frequencies.

4.3.1 Power and Performance Estimations based on PMCs

We use PMCs to estimate the power and performance of each of the different core modes and based on these values and the known frequencies of all the core modes, we compute the metric IPS^2/Watt. The PMCs chosen for our study are listed below.

1. ***IPC***: The longer the processor takes to execute an application, the more power it dissipates.

2. ***Cache activity***: Cache misses at any level in the hierarchy directly impact the performance and in turn, the power consumption. Therefore, the number of hits and misses at both Level 1 ($L1h$, $L1m$) and Level 2 ($L2m$, $L2h$) caches are important when estimating the power and performance.

3. ***Branch activity***: Branch mispredictions cause considerable loss in performance and power. Therefore, we track the number of *Branch mispredictions (Bmp)* .

4. ***Instructions committed***: Each instruction type (in the ISA) utilizes a separate set of resources. Thus, hardware counters which count the number of Integer (INT), Floating-point (FP), Load (L), Store (St), Branch (Br) instructions and the total number of *Fetched instructions (Fi)* are tracked.

5. ***Buffer-full stalls***: The performance of a processor suffers when the pipeline stalls due to lack of entries in the ROB, LSQ, IQ or RAT (register alias table).

4.3.1.1 Shortlisting Performance Counters

Our goal is to find the smallest set of counters that would allow us to estimate power and performance on each of the core modes with a reasonable accuracy. Monitoring fewer counters reduces the hardware overhead for estimation. PMC values are available only for the currently executing mode but we need to estimate power and performance for the other three core modes as well. For example, if the application is currently running on the average core mode, we need to estimate the power of this configuration and the power and performance for the other three configurations, namely the NC, LW and SM modes using the PMCs of the current AC mode.

To select the PMCs that exhibit the highest correlation to the required estimates (of power and performance) and then obtain the corresponding expressions (using linear regression) we select a training set of eight SPEC2006 benchmarks [11] which includes *sjeng, h264ref, soplex, omnetpp, bzip2, namd, gobmk, hmmer* where each of these benchmarks has application phases that prefer one of the four different core modes. The values of the counters listed previously were tracked at fine grain instruction granularity, i.e., after every 2K instructions committed during the execution of the benchmark. To select a suitable subset of counters, we use an iterative greedy algorithm based on the least squared error. The algorithm seeks to minimize the sum of squares of the differences between the estimated and actual power values. The correlation coefficient R^2 provides a statistical measure of how close the data are to

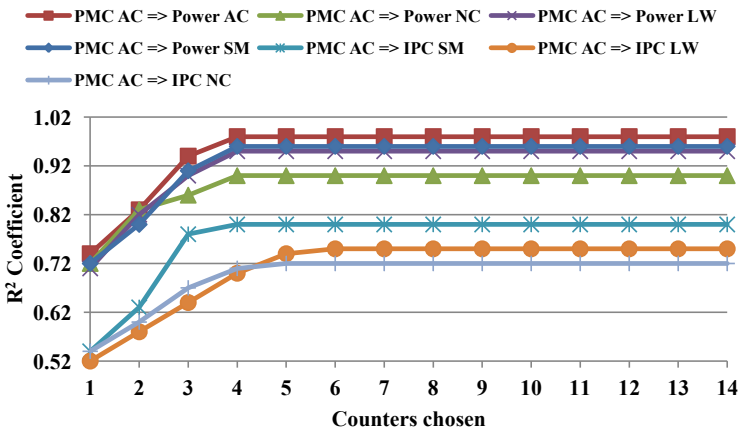

Figure 4.8. R^2 coefficient as a function of the number of chosen PMCs. PMC AC => Power NC denotes using the performance counters of the average core mode to estimate the power on the narrow core mode.

the fitted regression line. Starting with a set of counters, we iterate through all remaining counters to determine which among them is the best to add to the existing set. The candidate counter that yields the highest correlation coefficient is selected. Figure 4.8 shows the value of the coefficient R^2 when the PMCs of the average core mode (AC) are used to estimate the power and IPC on the other three core modes. The figure shows that four to five counters are sufficient as the R^2 value saturates afterwards. Note also that estimating the power in the same core yields a higher R^2 value than in other core modes indicating higher estimation accuracy. We need a minimum of four counters to estimate the IPC on the SM and NC core modes but R^2 saturates at five counters for the IPC of the LW core. Similar analysis was carried out for the estimations on the other three core modes. After selecting the most suitable counters, linear regression is used to derive the expressions for the performance and power estimation. Table 4.5 shows the expressions obtained for estimating the power and IPC for each of other three modes and the power of the AC mode using the values of the PMCs monitored while executing in the AC mode. For the sake of brevity, we show only the expression for estimating the power and IPC on each of other three

Table 4.5. Power (P) and performance (IPC) estimation for the other three modes using the performance counters values in the AC mode.

Estimated Parameter	Expression
AC ⇒ Power AC	$1.40{\cdot}10^{-2}{\cdot}\ L1h + 13.81 \cdot IPC$ $+2.95{\cdot}10^{-2}{\cdot}\ St - 1.18{\cdot}10^{-2}{\cdot}\ Bmp$ -0.29
AC ⇒ Power NC	$-1.30 \cdot Bmp - 0.85 \cdot L1m$ $+0.41 \cdot Br + 2.30{\cdot}10^{-2}{\cdot}\ St$ +0.46
AC ⇒ Power LW	$-0.34 \cdot L2m - 1.04 \cdot L$ -0.56 $\cdot Bmp - 1.40{\cdot}10^{-2}{\cdot}L1h$ +0.1
AC ⇒ Power SM	$-3.10 \cdot L1m + 6.67{\cdot}10^{-3}{\cdot}\ IPC$ $-4.20 \times 10^{-2} \cdot Bmp + 0.27$
AC ⇒ IPC SM	$0.21 \cdot L1h + 0.91 \cdot IPC$ $+0.11 \cdot L - 0.12 \cdot Bmp$ +4.46
AC ⇒ IPC LW	$0.12 \cdot IPC - 1.81 \cdot L1h$ $+0.31 \cdot St - 1.23 \cdot L1m$ +0.29
AC ⇒ IPC NC	$1.12{\cdot}10^{-1}{\cdot}\ Br + 1.81 \cdot IPC$ $+3.9{\cdot}10^{-2}{\cdot}\ St - 1.18{\cdot}10^{-2}{\cdot}\ L2h$ +0.38

core modes when using the PMCs of the average core mode. Similar expressions have been obtained for other combinations.

4.3.1.2 Accuracy of Power/Performance Estimation

The accuracy of the power and performance estimations is shown in Figure 4.9. The estimation error study was conduced for a set of 17 workloads that consists of mix of SPEC2000 and SPEC2006 benchmarks [11, 89]. We observe that the average error in estimating power is 8% which is significantly lower than the 16% average error in estimating IPC. Although the average estimation error is reasonably low, the actual estimation error may be considerably higher at some time instances and this may cause wrong morphing decisions. Therefore, we analyzed the temporal distribution of errors and the results are shown in Figure 4.10. This figure depicts the error in IPC estimation for the other three core modes using the PMCs of the NC core mode. We observe that the deviation of the errors from the mean is low for the majority of sample points with up to 80% of the sample points lie between ± 10% from the

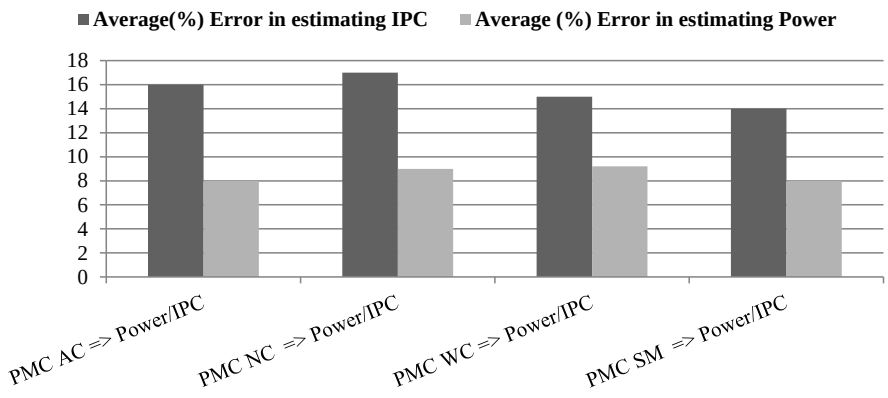

Figure 4.9. Average error in estimating power and IPC in all core modes using the PMCs on the current mode. E.g., PMC AC ⇒ Power/IPC denotes the average error in estimating power and IPC in all other core modes using the PMCs of the average core (AC) mode.

mean. This demonstrates that the average error is a sufficiently good indicator for the instantaneous estimation error. In our experiments we have observed very few decision errors.

4.3.2 Morphing Controller

To enable morphing between different core modes, we need an on-chip controller that governs all the required changes in the core configuration upon morphing. We envision this controller to be a variant of similar controllers for core morphing [61]. It obtains periodically PMC values from the current core configuration and estimates the IPS^2/Watt for the current and alternative core modes. The estimation is based on the expressions described previously. We assume that the controller includes a multiply and accumulate (MAC) unit that is pipelined and capable of completing 1 MAC operation per cycle, and is power gated when not in use. The IPS^2/Watt estimates determine the best mode to morph into. If the estimated IPS^2/Watt for one of the modes is sufficiently higher than for the current mode, a mode transition is initiated. Mode switches incur some overhead especially if a change in voltage/frequency is

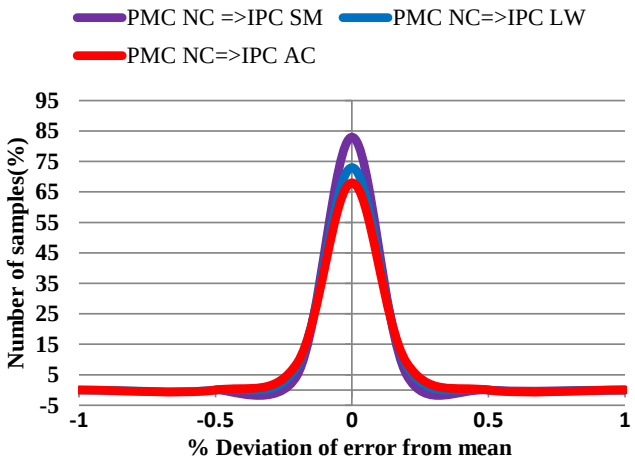

Figure 4.10. Estimation error distribution when using the PMCs of the NC mode to estimate the IPC for the remaining three core modes.

required as the PLL must be relocked to the new operating conditions. The overheads imposed by the controller are discussed in the next section. The controller sets the voltage and frequency by placing values in the Voltage Control Register (VCR) and the Frequency Control Register (FCR). The Voltage Regulator Module (VRM) reads the VCR and sets up the new voltage. Similarly, the FCR controls the frequency division within the PLL. Finally, the controller also features a Configuration Control Register (CCR) that directs which units should be powered on or off.

4.4 Experimental Setup

4.4.1 Simulator and Benchmarks

To evaluate our proposed morphable core architecture we have used Gem5 as a cycle accurate simulator with integrated McPAT modeling framework to compute the power of the core and L1 caches [10, 59]. We ran experiments using 17 benchmarks from the SPEC2006 and SPEC2000 benchmarks suites [11, 89]. The benchmarks were cross compiled using gcc for Alpha ISA with -O2 optimization. In the simulation

experiments we executed 4 billion instructions of each benchmark after skipping the first two billion.

4.4.2 Determining the Window Size

Power and performance estimates are calculated after a fixed number of committed instructions referred to as *window*. To prevent switching too frequently (e.g., after every window) we wait until the particular phase of the currently executing application has stabilized. To this end, we wait for a fixed number of windows to elapse before making a decision to switch modes. We term this number of windows the *history depth*. A decision to switch modes is then made based on the most frequently recommended core mode during the windows in this period. This way, short periods of transient behavior of the executing application will not result in a core mode switch.

We denote by n the total number of retired instructions during this period where $n = \textit{history depth} \times \textit{window length}$. For example, if for the past n committed instructions, moving from the average core to the narrow core mode was the most frequent recommendation, we conclude that the application has entered a phase where the narrow core may provide a higher IPS^2/Watt and we switch from the average to the narrow core mode. We have conducted a sensitivity study to quantify the impact of the *window length* and *history depth* on the achieved benefits. The *window size* and *history depth* combination that yields the highest IPS^2/Watt for the entire program execution would be the best choice. The window length was varied from 250 to 1000 instructions in steps of 250. Within a particular window, the history depth was varied from 1 to 10. For example, a window length of 500 and history depth of 4 means that we make a reconfiguration decision at the end of every 2K instruction (500×4). Along with varying window length and history depth, we iteratively run experiments to determine the minimum improvement IPS^2/Watt threshold that determines the

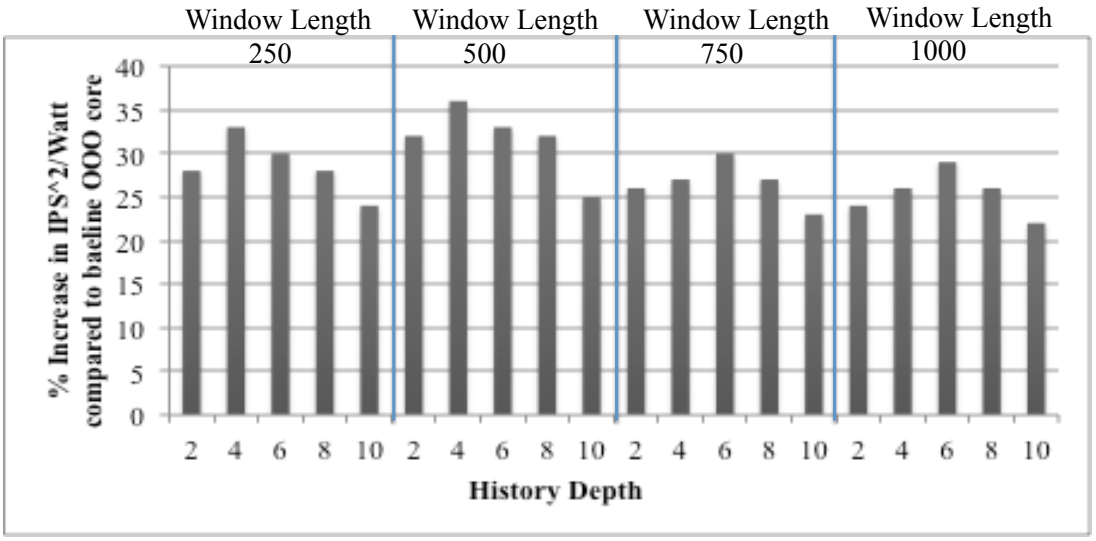

Figure 4.11. Percentage increase in IPS^2/Watt over AC core-mode for a range of window lengths and history depth

mode switch. It was determined to be 5%. Figure 4.11 shows the achieved increase in the average IPS^2/Watt when switching to the preferred core mode from the current AC mode for the SPEC benchmarks. Based on this figure, a window length of 500 and history depth of four provide the largest improvement in IPS^2/Watt. Thus, in all our remaining experiments, a reconfiguration decision is done at the end of every 2K instructions.

4.4.3 Morphing overhead

In the proposed scheme, the voltage and frequency may change at a fine instruction granularity. The potential overhead of frequent voltage and frequency scaling must be taken into account. Traditionally, DVFS has been applied at coarse instruction granularity, of the order of millions of processor cycles, due to high overhead that is involved in scaling voltage and frequency using an off-chip regulator [72]. Recently, Kim *et al.* has proposed the use of an on-chip regulator which reduces the time needed for scaling voltage to tens of nanoseconds or hundreds of processor cycles [49]. Using an on chip regulator, a low overhead (hundreds of cycles) DVFS can be performed

at a fine grain instruction interval. A hardware-based fine grain DVFS mechanism that uses an on chip regulator was implemented by Eyerman *et al.* where DVFS was performed upon individual off-chip memory accesses [29]. We assume that such an on chip regulator has been included in the processor design. The authors of [49] have estimated the DVFS latency to be 200 cycles. In our experiments we have used this 200 cycles DVFS latency that constitutes a major component of the overall core morphing overhead. As the latter is design dependent, the result section includes analysis of the impact of higher overheads on the core performance.

Overheads associated with power-gating/power-up of banks of ROB, LSQ, IQ and partial powering on/off of fetch and decode units are also taken into account. When power gating individual units/banks, no dynamic energy is consumed and the static energy consumed by these idle units is low. Power-gating/power-on of all the blocks simultaneously may lead to a sudden power surge and therefore, we assume staggered power gating where only a single bank is gated in a given clock cycle. Powering off a single bank is expected to take tens of clock cycles [27]. The bank selected to be turned off is the one with the smallest number of used entries. If the selected bank is not empty we must wait until all its entries are vacated before switching it off.

Calculating IPS^2/Watt using the PMC-based performance and power estimates involves a computational overhead. To compute IPS^2/Watt, 7 expressions (shown in Table 4.5) must be evaluated online, which require four MAC operations per expression. The resulting computation overhead is about 30 clock cycles. Once the mode switch decision is made, the controller needs to set the voltage and frequency registers with new values and initiate a mode switch which incurs an additional overhead. Taking into account all the individual overheads we, conservatively, estimate the total overhead to be 500 cycles. As the frequency of core reconfiguration is not high (as will be shown in the next section), even a higher morphing overhead will have a negligible

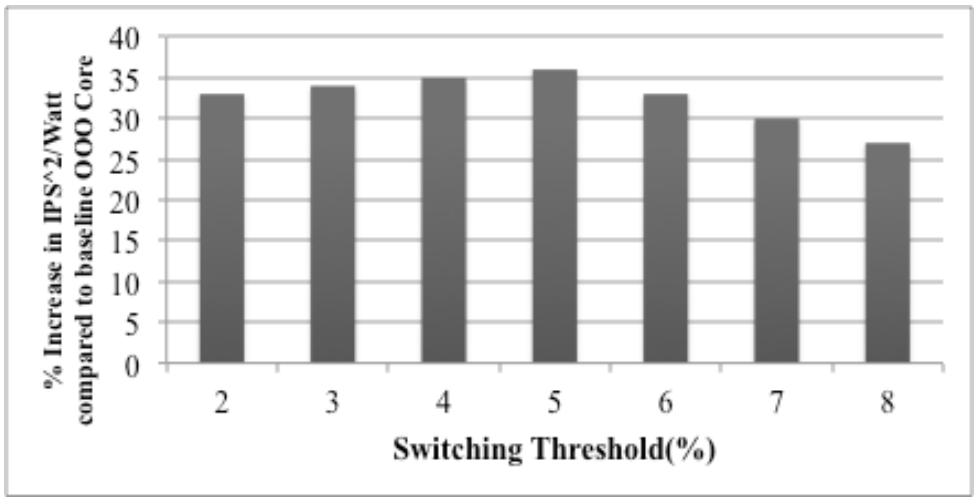

Figure 4.12. % Increase in IPS^2/Watt vs various switching threshold

impact. We analyze in the next section the impact of a higher overhead on the core performance.

4.5 Evaluation

4.5.1 Power Efficiency

As mentioned previously, applications exhibit diverse phase behavior and the core mode on which an application runs most efficiently changes during the course of execution. The decision to switch modes is based on the metric IPS^2/Watt that assigns higher significance to performance than to power.

The IPS^2/Watt metric is calculated using the expressions for estimating the power and performance (see Table 4.5). To avoid frequent switching, we have done sensitivity analysis to determine the right switching threshold as shown in Figure 4.12. At lower threshold (2%), reconfiguration happen too frequently for insignificant gains in IPS^2/Watt, thus increasing the reconfiguration overhead resulting in reduced IPS^2/Watt improvements. Beyond 6%, there is reduced benefit due to reduction in number of reconfigurations. The IPS^2/Watt threshold is set therefore to 5%.

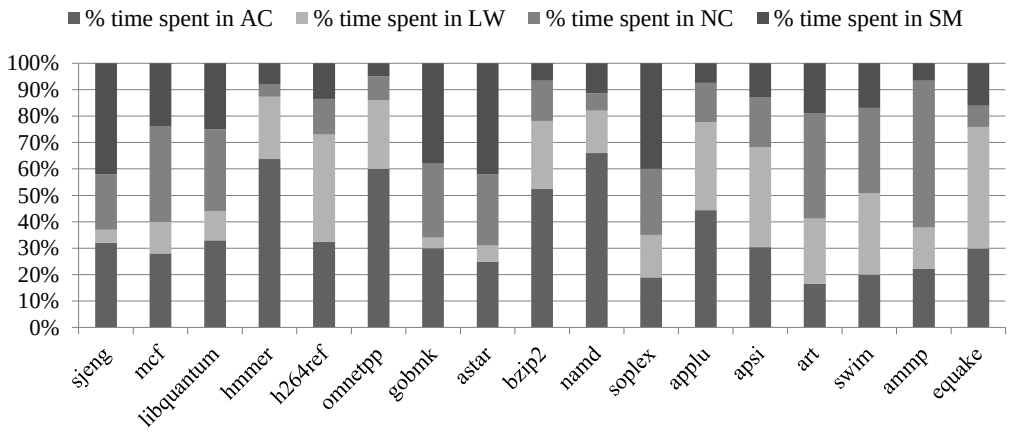

Figure 4.13. Tenancy of core modes for the unconstrained power core.

Figure 4.13 shows the percentage occupancy in each of the four core modes in the unconstrained power case. The figure demonstrates the diversity in the use of four core modes by the different benchmarks and also shows that each of the four modes is highly utilized (more than 40% of the time) in some of the benchmarks. The morphable architecture presented in [61] consists of two core modes (OOO and InO) and only benefits applications that have memory intensive phases or phases with high branch mis-prediction rates, as these phases are mapped to the power efficient InO core. Compute intensive benchmarks do not benefit as much from the two-mode morphable architecture as they have very few phases with low performance that could be mapped to an InO core. Our proposed morphable architecture caters to more diverse application phases due to the four distinct core modes that relieve diverse resource bottlenecks.

The percentage improvement in IPS^2/Watt for the SPEC benchmarks executing on our proposed morphable architecture when compared to executing completely on the AC core mode is shown in Figure 4.14. On an average (using geometric mean), we achieve an IPS^2/Watt improvement of 37% compared to the baseline architecture of the AC core. Benchmarks which are memory intensive or have high branch mis-prediction rates, such as *mcf, soplex*, and *astar*, achieve larger IPS^2/Watt im-

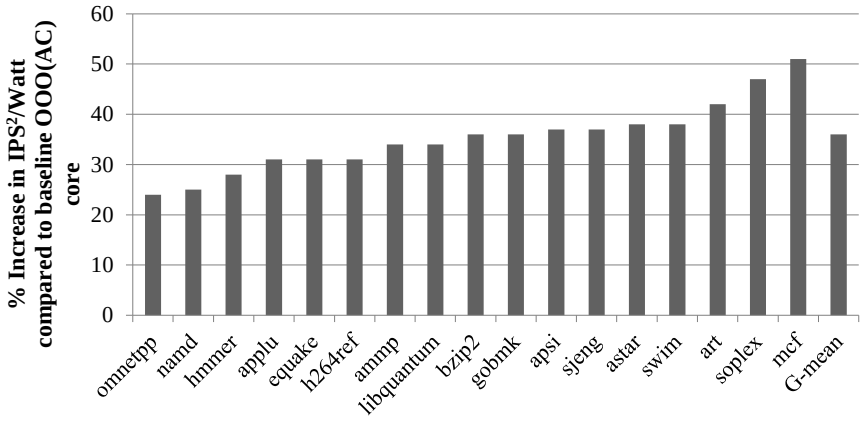

Figure 4.14. IPS^2/Watt improvement of the proposed morphing scheme compared to execution on AC mode for SPEC benchmarks.

provements since they can be mapped to an energy efficient core mode. Compute intensive benchmarks, such as *hmmer, bzip2,* and *h264*, also take advantage of the proposed morphable architecture. We observe on average of 34% improvement in IPS^2/Watt for the compute intensive benchmarks compared to the 38% improvement for memory intensive ones.

The morphable core should allow a wide variety of applications to run effectively on different core modes. We ran benchmarks from the MiBench[37] and Mediabench [58] suites to test our morphable core capabilities on applications apart from SPEC benchmarks. As seen from Figure 4.15, we obtain on average 15% IPS^2/Watt improvement demonstrating the benefits of the morphable cores for a wide range of applications. Note however, that the MiBench and Mediabench applications achieve a lower benefit as they do not have as diverse program phases as the SPEC benchmarks.

4.5.2 Comparison to Other Switching Schemes

We compare our PMC-based fine-grain core mode switching scheme, referred to as *FineGrain_PMC*, to three other switching schemes, namely: (i) Sampling based

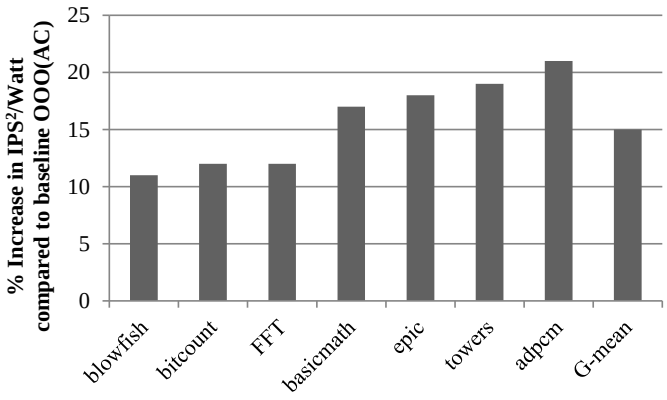

Figure 4.15. IPS^2/Watt improvement of the proposed morphing scheme compared to execution on AC mode for Mediabench/Mibench benchmarks.

switching within a morphable architecture, referred to as *CoarseGrain_sampling*; (*ii*) Oracular scheme referred to as *Oracular_Switch*; and (*iii*) PMC-based switching at coarse grain granularity, referred to as *CoarseGrain_PMC*.

To this end, we have implemented the morphable architecture presented in this thesis with morphing decisions made based on sampling. The parameters used for this implementation include a switching interval of 1M instructions and a sampling interval of 10K instructions [69]. We have also implemented the oracular scheme where an oracle determines, every 2K instruction, which is the best core mode for the next interval of 2K instructions. The third implemented scheme is a PMC-based one making switching decisions at a coarse grain granularity of 1M instructions.

Figure 4.16 compares the IPS^2/Watt and the energy savings obtained for the four switching schemes. The *CoarseGrain_sampling* yields 14% less energy savings than the *FineGrain_PMC*. The reason for lower energy savings for the sampling-based scheme is twofold. First, sampling is wasteful when the program is already running on the best available core. Second, sampling is performed at a coarse grain level thus missing opportunities available at finer granularity. Thus, our PMC-based run-time decision mechanism helps in making the right morphing decisions yielding higher en-

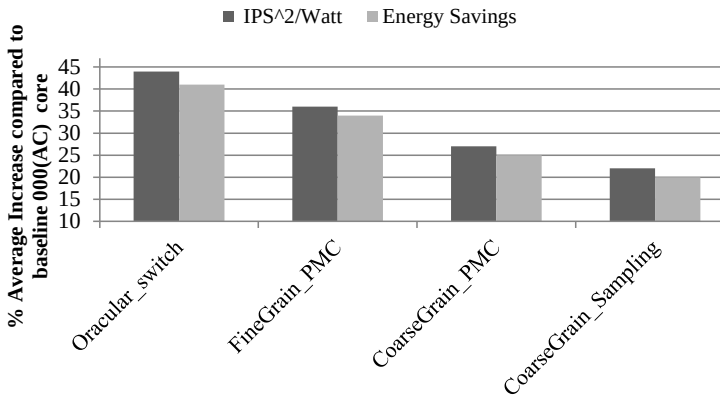

Figure 4.16. Comparing the IPS^2/Watt and energy saving of four morphing schemes.

ergy savings. We also compare the IPS^2/Watt of the coarse grain and fine grain PMC-based schemes. *FineGrain_PMC* yields 11% higher IPS^2/Watt compared to *CoarseGrain_PMC*. This scheme also does much better than *CoarseGrain_sampling* due to a smaller performance overhead in PMC schemes compared to sampling based ones. The oracular scheme achieves a higher IPS^2/Watt, by 10%, than our *FineGrain_PMC* scheme. As the oracular scheme cannot be implemented in practice, it provides an upper-bound for the maximum IPS^2/Watt that could potentially be achieved by our approach.

Figure 4.17 compares the IPC improvements over the baseline average core for the four switching schemes. The IPC value obtained for the power budget of 2W and 1.5W is normalized to that of the corresponding average core (AC) mode obtained with 2W and 1.5W power constraint, respectively. For the unconstrained case, we observe a 9% improvement in IPC over the baseline (AC) core mode using the *FineGrain_PMC* scheme compared to the 3% achieved by the sampling-based scheme. The oracular scheme shows an upper bound of 12% IPC improvement. For a 2W power budget, a 7% improvement in IPC is achieved by the *FineGrain_PMC* scheme compared to 2.5% for the sampling-based scheme. These results show that our morphing scheme improves performance although its main goal is to improve the performance/Watt.

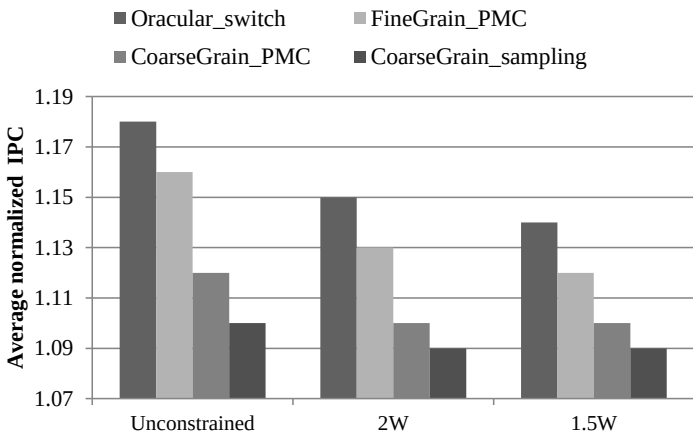

Figure 4.17. IPC comparison for power constrained and unconstrained cores for various switching schemes.

Figure 4.18 shows the reduced performance improvement experienced by the different morphing schemes for increasing values of the core morphing overhead. Although our initial estimated cost (overhead) of morphing is 500 cycles but the actual overhead is calculated during the simulation accounting for draining of banked resources. As mentioned previously, the morphing overhead is design dependent and thus it is important to estimate the impact of a higher overhead. Figure 4.18 shows that as the overhead increases from 500 to 1K cycles, the performance drops by 3.5% for our *FineGrain_PMC* scheme. Higher increases in the morphing overhead result in larger performance losses indicating that switching at fine granularity must have a fast switching mechanism.

Figure 4.19 shows the number of switches in our 4-mode morphable architecture at various instruction granularities. As expected, a greater number of reconfigurations takes place at lower instruction granularities, thus yielding higher IPS^2/Watt when compared to coarse grain switching at instruction granularity of 10K and above. For our selected 2K instruction interval the number of switches on average is 12500 in 100M instructions, i.e., after every 2K instructions we have a probability of 25% to perform a mode switch.

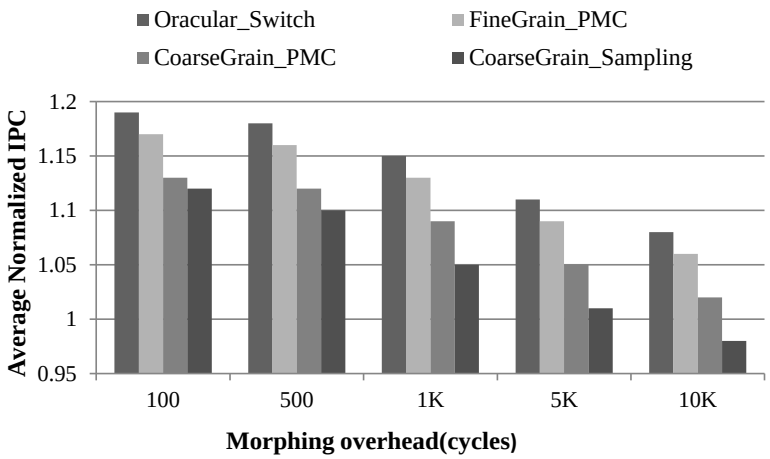

Figure 4.18. The impact of morphing overhead on IPC.

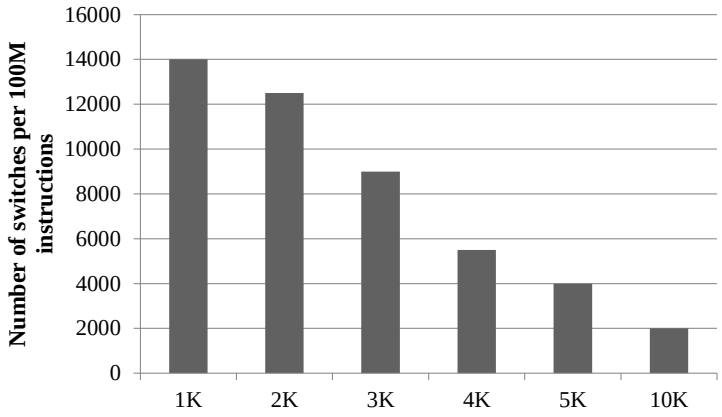

Figure 4.19. Number of switches per 100 million instructions for a range of instruction granularities for the power unconstrained 4-mode morphing scheme.

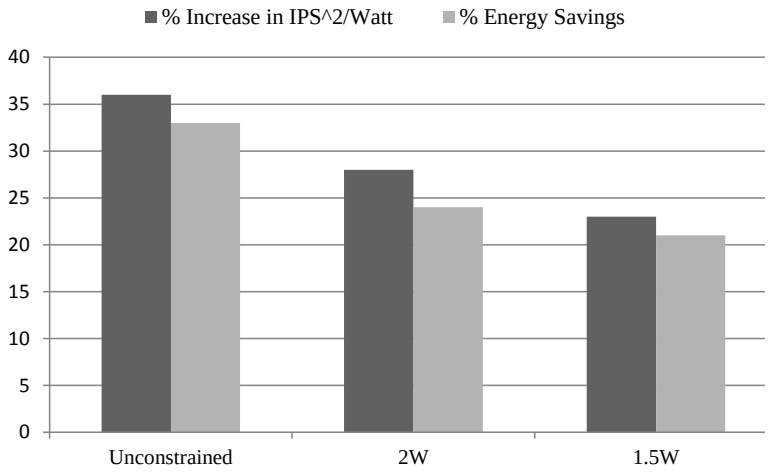

Figure 4.20. IPS^2/Watt and energy savings for the power constrained and unconstrained cases compared to execution in baseline OOO(AC) core.

Figure 4.20 shows the impact of power constraints on the IPS^2/Watt improvements and the energy savings achieved by our morphing scheme. For the unconstrained power case, we obtain a 36% IPS^2/Watt improvement and 33% energy savings (compared to the average core mode), while for the 2W power budget case, the IPS^2/Watt improvement is only 27% and the energy savings drop down to 24%. We also compare the throughput/Watt achieved by non-monotonic architecture using *FineGrain_PMC* with the coarse grain switching with *FineGrain_PMC* architecture. We obtain 12% more throughput/Watt using *FineGrain_PMC* architecture compared to NM architecture designed with coarse grain switching support.

4.5.3 Comparison of the 4-mode Morphable Core to the Big/Little architecture

To compare our 4-mode morphable core to the previously proposed 2-mode architectures (OOO and InO) [47, 61] we analyzed an OOO/InO morphable architecture proposed in Chapter 1. Whenever a decision is made to switch from OOO to InO or vice-versa, the fetch width is reduced, half the decoders are powered off, some of the functional units are shut down (e.g., INT ALUs reduced from 4 to 2, FP ALUs

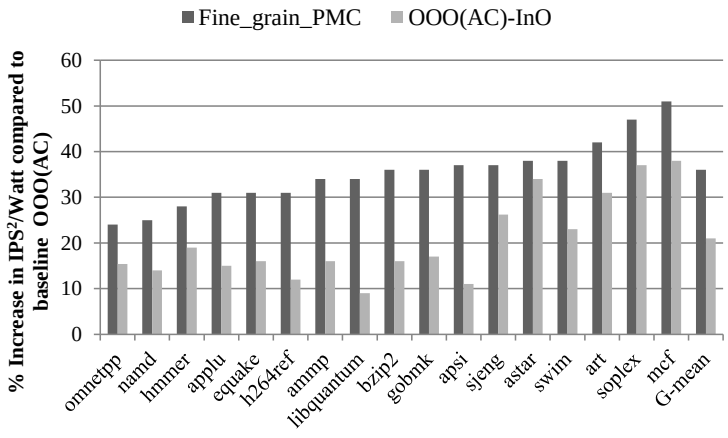

Figure 4.21. Comparison of the IPS^2/Watt improvement (over execution on the AC mode) between our *FineGrain_PMC* morphable core and the 2-mode morphable core (AC,InO).

reduced from 2 to 1) and the ROB and Register Alias Table (RAT) are powered off. Turning off structures (while moving to InO mode) by clock gating was employed in [47]. When a mode switch happens, the pipeline is drained and several units are powered on/off depending on the core mode we are morphing into, and then instruction execution starts in the new mode.

The selected architectural parameters for the OOO core are those of the AC mode in Table 4.2. The InO core has a fetch/issue width of 2, IQ with 36 entries, and cache sizes and frequency identical to those of the AC core. The instruction granularity at which core switching decisions are made was set to 2K instructions. The decision to morph is based on the estimated IPS^2/Watt using a PMC-based estimation mechanism. The IPS^2/Watt improvement achieved by the 2-mode morphable core (AC,InO) (over executing on the AC mode) is compared to that achieved by our 4-mode core in Figure 4.21. On average, the 2-mode core achieves a 21% increase in IPS^2/Watt versus the 37% achieved by our 4-mode core. Figure 4.22 compares the average increase in IPS^2/Watt, IPS/Watt and energy savings achieved by our 4-mode *FineGrain_PMC* with those obtained when using the 2-mode (OOO(AC),InO)

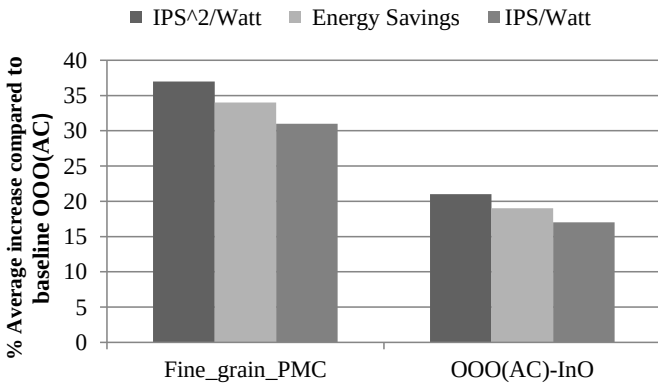

Figure 4.22. Comparing the IPS^2/Watt, IPS/Watt and energy savings between the *FineGrain_PMC* and the 2-mode (AC,InO) morphable core.

scheme. On average, the 4-mode scheme achieves a 12% higher IPS/Watt and a 14% higher energy saving compared to 2-mode (OOO(AC),InO) scheme.

The goal of the next experiment is to determine whether including an InO mode is necessary or it can be replaced by our SM mode that is still an OOO core but has a width of 1 and minimal sizes of ROB and other buffers. Figure 4.23 compares two schemes: the first one has two core modes, OOO (AC) and OOO (SM) while the second has three core modes, namely, OOO (AC), OOO (SM) and InO. We observe that the 3-mode morphing scheme that includes an InO mode provides an additional 6% IPS^2/Watt improvement over the simpler 2-mode morphing. We conclude that the inclusion of the InO core does not sufficiently improve IPS^2/Watt to justify the increased design complexity of supporting the two very different core architecture styles, i.e., OOO and InO. We have, therefore, excluded the InO mode to keep the micro-architecture simple.

4.5.4 Benchmark Analysis

In this section we focus on the characteristics of different benchmarks and try to understand why some benchmarks prefer one mode of the morphable architecture

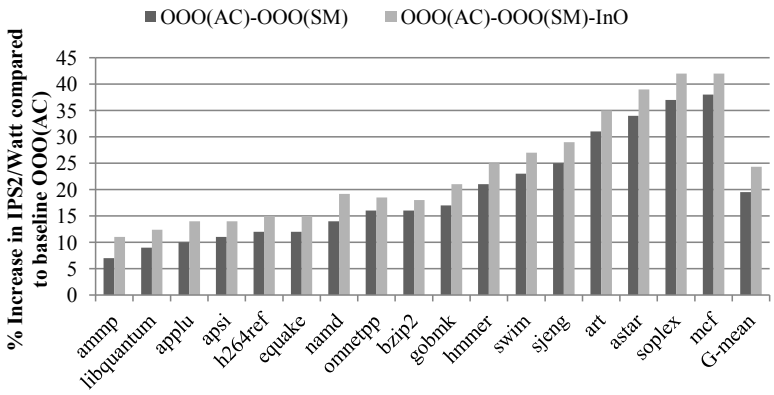

Figure 4.23. Comparing the improvement in IPS^2/Watt between a 2-mode morphable core (AC, SM) and a 3-mode morphable core (AC, SM, InO).

over the others. The characteristics that we study include branch mis-predictions, occupancy of buffers (LSQ, IQ, ROB), L2 cache misses and IPC. Benchmarks with high branch mis-prediction rates have low ILPs and are not expected to benefit from a higher frequency. Such benchmarks would therefore, prefer the small core mode (SM) that runs at reduced frequency and has small resource sizes. To illustrate this we show in Figure 4.24 the temporal behavior of the benchmark *astar*, that has high branch mis-prediction rates, and compare its performance while running on the SM and AC core modes. During this period of program execution *astar* exhibits a high branch mis-prediction rate and as a result, the IPC difference between the AC and SM modes is small but executing in the SM mode improves the IPS^2/Watt.

Memory-bound applications, e.g., *libquantum, mcf* and *xalancbmk*, experience a large number of L2 misses and generate many parallel loads. Thus, these benchmarks prefer running in the narrow (NC) mode which has a higher frequency and reduced buffer sizes. The higher frequency helps when many independent loads are invoked. A similar observation was reported in [69]. Figure 4.25 shows a portion of the behavior, as a function of time, of the benchmark *mcf* and compares its IPC when running on the NC and AC modes. The section of program shown in Figure 4.25 has a high

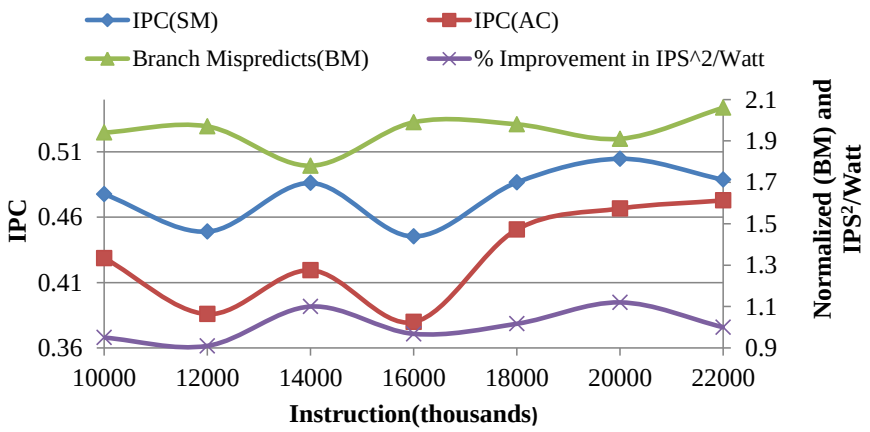

Figure 4.24. Analysis of the benchmark *astar* at a fine instruction granularity, comparing its execution in the SM and AC modes.

L2 miss rate. The L2 misses were monitored while running on the AC mode. When the L2 miss rate is high, the NC core mode provides a higher IPC than the AC mode since its higher frequency helps in issuing independent loads. The performance difference between the two core modes is small for low L2 miss rates. The IPS^2/Watt is improved by up to 7% by running in the NC rather than the AC mode.

Compute-bound applications, like *bzip2, hmmer* and *h264ref*, have high IPC and their performance is limited by issue width and buffer resources and not by L2 cache misses or branch miss predictions. Therefore, these benchmarks tend to prefer the Large Window (LW) core mode. Figure 4.26 compares the execution of the *bzip2* benchmark in the LW and AC modes. The number of times when one of the buffers, ROB or LSQ or IQ, became full while running in the AC mode is also shown in the figure. We observe that providing larger resources alleviates the problem of buffers getting full and improves the IPC. Improved IPC and reduced frequency while running in the LW mode, compared to the AC mode, provides an IPS^2/Watt improvement of up to 6%.

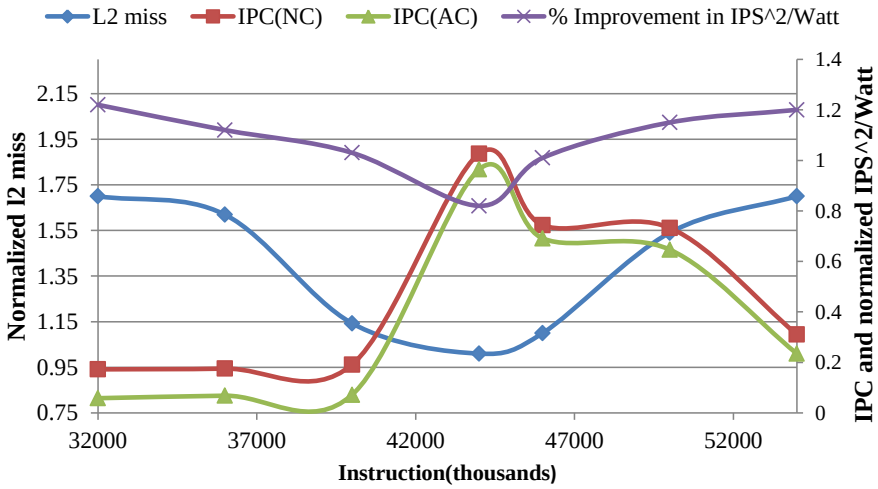

Figure 4.25. Analysis of the benchmark *mcf* at a fine instruction granularity, comparing its execution in the NC and AC modes.

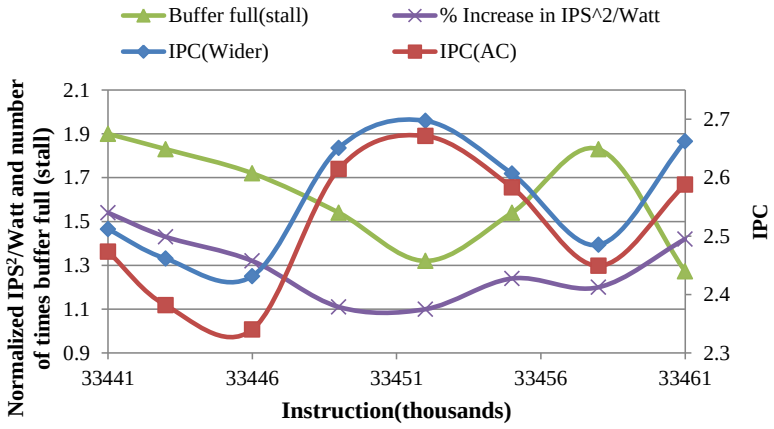

Figure 4.26. Analysis of the benchmark *bzip2* at a fine instruction granularity, comparing its execution in the LW and AC modes.

4.6 Conclusion

In this chapter we proposed a morphable core design that can assume one of four different core modes. Apart from the baseline average core mode, the additional core modes are suited to address most common performance bottlenecks found in the considered benchmarks. Based on a small number of performance counters, a novel runtime mechanism estimates the performance and power across all core modes and uses this information to determine the core mode that offers the best power efficiency. The cache was not resized across core modes to support fast switching from one mode to another enabling fine-grain morphing. We have shown that the proposed four-mode morphing offers higher power efficiency than the two-mode morphing considered earlier. It was also shown that fine-grain switching between core modes outperforms switching at a large instruction granularity which misses power saving opportunities. Our results indicate that the four-mode morphable core achieves an IPS^2/Watt gain of 37% compared to a standard OOO core and 16% higher energy efficiency compared to big/little morphable architectures. Importantly, unlike previous self-morphing schemes that only improves throughput/power but not performance, we improve performance by 9%.

CHAPTER 5

ON-LINE RECONFIGURATION VS DYNAMIC VOLTAGE AND FREQUENCY SCALING (DVFS)

In the previous chapters, we have explored reconfigurable and morphable architectures that adapt to changing resource demands of workloads and enable fast switching by reconfiguring into various core modes with varying core resource sizes, voltage and frequency. Many current processors employ DVFS aggressively to either improve energy efficiency or increase performance [20]. For example, memory-bound phases of an application might not have sufficient ILP to keep the core busy, providing opportunities for scaling down the voltage and frequency. Such voltage/frequency reduction provides a cubic reduction in power with limited performance loss [44]. Intel's turbo-boost technology increases the frequency of active cores when other cores are idle, providing enhanced performance [20]. In the past, DVFS involved high overhead (tens of microseconds) as it relied on an off-chip voltage regulator to switch from one voltage to another, and consequently, DVFS was performed at coarse grain OS switching granularity of millions of instructions [49]. Kim et al. have shown that if on- chip voltage regulator is used, the voltage switching time for DVFS is reduced significantly to the order of nanoseconds enabling fine grain voltage scaling [49]. Recently, Intel introduced a fully integrated voltage regulator (FIVR) in their Haswell processor, reducing the voltage transition time, thus enabling voltage/frequency scaling at a finer granularity [17]. The integrated on-chip regulator is fast, offers minimal parasitic losses with a reduction in the PCB footprint but an increase in chip area. Fine grain DVFS and reconfigurable architectures provides two alternatives for improving processor energy efficiency. In the previous sections, we have evaluated the power

Table 5.1. Voltage and Frequency levels considered.

Frequency (GHz)	Voltage (V)
1.2	0.8
1.4	0.8
1.6	0.9
1.8	1.0
2	1.1

efficiency benefits of reconfigurable architecture at fine granularity. We have shown that combining DVFS with resource resizing can provide improved energy efficiency. Lukefahr et al. compared the energy efficiency of the Big/Little reconfigurable architecture to that of fine grain DVFS concluding that the Big/little reconfigurable architecture can provide a higher energy efficiency than fine grain DVFS only [60]. In this work, we extend this comparison to a more complex (than Big/little) AMP and we study the power efficiency achieved by various architectures as listed below:

1. We study the power efficiency of DVFS and reconfigurable architectures for single-threaded application at fine grain switching frequency and answer the question whether fine-grain reconfigurable architectures provide higher power efficiency than fine-grain DVFS only.

2. We study the effect of using on-chip and off-chip regulators for DVFS and compare to a reconfigurable architecture that uses an on-chip regulator.

3. We evaluate the DVFS scheme based on PMC-based online scheme and compare the efficiency of this scheme with oracular approach.

5.1 Evaluation Framework

We plan to compare the performance/Watt of architectures consisting of architectures that employ only DVFS on the cores and architectures that support dynamic core reconfigurations.

Fine grain DVFS (Fine_DVFS): The base core for fine grain DVFS simulation experiments to be used, is the AC core shown in Table 4.2 of Chapter 4. Note that in these experiments, only the voltage and frequency are varied; the resources stay constant. The frequency and voltage combinations that will be used for these experiments are shown in Table 4.2. Achievable frequencies for given voltage levels are chosen from [1]. The overhead for fine grain DVFS (at 2K instructions granularity) is assumed to be $100ns$, based on [29, 49].

Coarse grain DVFS (Coarse_DVFS): In this scheme, the core configuration is the same as above but the switching granularity is 1M instructions due to the higher overhead of an off-chip regulator that is assumed to be $50\mu s$ based on [29, 49].

NMRA: This is similar to what was described in Chapter 4, which consisted of four core modes as shown in Table 4.2. The instruction granularities are 2K for *NMRA_Fine* and 1M for *NMRA_Coarse*.

MRA: This is similar to what was described in Chapter 2 and includes two core modes. Switching is done at a 2K instructions granularity. It consists of an OOO core that is similar to the AC core in Table 4.2 and an InO core.

Big_Little: This architecture resembles ARM's Big_Little [35]. It differs from the MRA architecture where the OOO and InO cores are separate cores with each having its own L1 caches and a shared L2 cache. The OOO core is the baseline Average Core (AC). Switching between the cores is performed at coarse grain granularity of 10M instructions. Migrating task from Big core to a small core incurs an overhead of 20K cycles [35].

5.2 Runtime Mode Selection

An effective runtime management is necessary for the application to run on the most suitable core mode or choose the most appropriate voltage/frequency at runtime (for architectures that employ only DVFS). All the architectures considered in this

work use the same runtime scheme, explained below, to make on-line reconfiguration decisions. We employ the PMC-based runtime scheme described in prior chapters, to effectively map the application to the appropriate mode [80, 90] .

5.2.1 Decision Metric

The decision metric chosen for selecting a new mode is IPS^2/Watt [36, 5] as it assigns a higher weight to performance than power. The mode that is estimated to provide the highest IPS^2/Watt is the one that will be used in the next time interval. To calculate the value of IPS^2/Watt for the different modes, we wish to use as few PMCs as possible to accurately estimate (at runtime) the power and performance.

5.2.2 Performance and Power Estimation using PMCs

The PMCs selected for computing the power and performance should obviously, have good correlation with power and performance and limited mutual correlation. Once the most suitable PMCs are identified, a linear regression is used to derive expressions for power and performance. To make mode switch decisions, we need to use the values of the PMCs in the currently executing mode to estimate IPS^2/Watt in each of the other modes. As shown previously, using PMCs, power and performance can be accurately estimated not only in the current mode but also in the other modes To derive the linear regression expression for power and performance, we use a set of training workloads which are chosen to have diverse application behavior. Power and performance equations are then derived individually for each of the architectures explored in this work. Figure 5.1 shows the accuracy in estimating the average power and IPC across different architectures as explained previously. For example, the NMRA $\rightarrow$ Power/IPC in Figure 5.1 show that, power and IPC across different core modes in the NMRA architecture can be estimated with an accuracy of 8% and 14%, respectively. Fine grain schemes such as (Fine_DVFS, NMRA, MRA) provide higher estimation accuracy than coarse grain scheme (Coarse_DVFS, Big_Little).

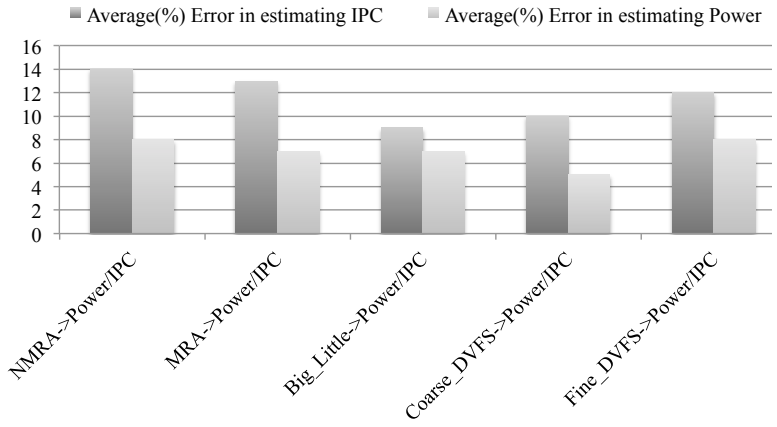

Figure 5.1. Average % error in estimating Power and IPC across different modes in each of the different architectures.

Thus, PMC-based estimation scheme can estimate power and performance on-line with reasonably high accuracy across different architectures.

5.3 Results

5.3.1 Power Efficiency Evaluation

For each of the schemes, the results are normalized with respect to the baseline Average core (AC) shown in Table 4.2. The PMC-based runtime scheme is used for each of the different architecture.

5.3.1.1 Fine/Coarse DVFS vs Fine/Coarse NMRA schemes

We compare the throughput/Watt for 4 different architecture schemes as shown in Figure 5.2. The figure shows that the Fine_DVFS scheme provides 8% more throughput/Watt compared to the Coarse_DVFS scheme due to its ability to transition between different V/F levels at a lower overhead. The NMRA_Fine scheme consists of several core modes running at different V/F to better match the diverse application behavior. The NMRA_Fine scheme provides 16% more throughput/Watt compared to the Fine_DVFS scheme by adapting to the resource demands of the application phases online through core resource resizing and DVFS. The NRMA_Fine

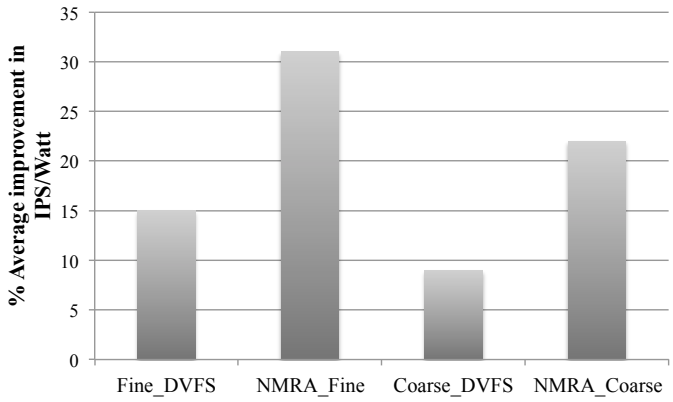

Figure 5.2. Throughput/Watt comparison between DVFS and NMRA architectures.

architecture captures diverse short phase behavior at finer granularity such as low performance phases, branch intensive phases, memory intensive phases and low ILP phases. The Fine_DVFS provides its highest benefit when targeting memory intensive phases (stalling on L2 misses). In contrast, the NRMA_Fine architecture adapts to various workload behaviors, resulting in a higher throughput/Watt than the Fine_DVFS scheme. The NRMA_Coarse scheme achieves 11% less throughput/Watt compared to the NRMA_Fine architecture due to coarse grain switching, thus missing out on fine grain opportunities for improving power efficiency. We conclude, therefore, that NRMA_Fine architectures provide a higher throughput/Watt than Fine/Coarse DVFS architectures.

Figure 5.3 compares the throughput/Watt between Fine_DVFS and NMRA_Fine for different flavors of benchmarks. Computation intensive benchmarks such as *bzip2, hmmer, namd, h264ref* do not benefit significantly from DVFS. Memory intensive benchmark such as *mcf, libquantum, soplex, swim* have a high L2 miss and independent loads and can take advantage of DVFS and achieve higher throughput/Watt. Branch intensive benchmark such as *astar, sjeng* also use DVFS during periods of high branch miss-predictions to improve throughput/Watt. NMRA_Fine uses DVFS

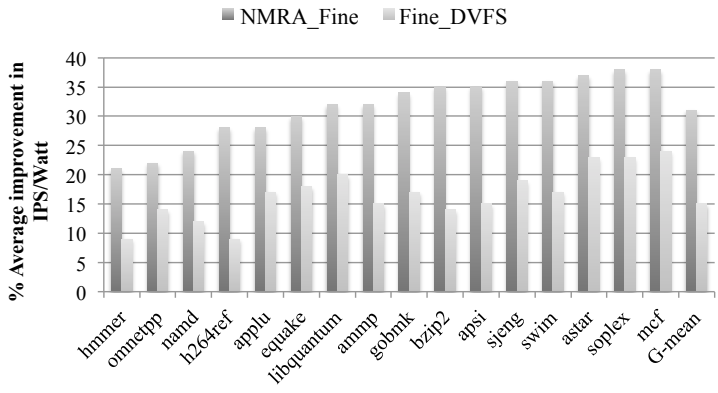

Figure 5.3. Throughput/Watt comparison between Fine_DVFS and NMRA_Fine architectures.

along with resource resizing to provide higher throughput/Watt for different flavors of benchmarks. Branch intensive benchmarks that have low ILP, make use of the SM core mode which has a lower frequency and reduced resource sizes, to provide higher throughput/Watt than the Fine_DVFS scheme. Computation intensive benchmarks are limited by resource sizes and issue width, they make use of the LW core mode to improve the throughput/Watt. Memory bound benchmarks make use of the NC core mode that has reduced resources and higher frequency which helps in accelerating independent loads.

5.3.1.2 Fine DVFS vs NMRA vs Fine/Coarse BigLittle architectures

Figure 5.4 compares four different architecture schemes. Along with NMRA_Fine and Fine_DVFS, we also compare architectures that switch only between OOO and InO core. As mentioned previously, the MRA scheme reconfigures an OOO into an InO core at fine granularity, whereas the Big_Little architecture switches between separate OOO and InO core types at coarse granularity. The NMRA_Fine scheme provides 11% higher throughput/Watt compared to the MRA scheme. The MRA scheme only switches between two very different core modes (OOO and InO) and thus does not cater to the diverse demands of applications as the NMRA scheme

does. The Big_Little architecture achieves 6% less throughput/Watt compared to the MRA scheme. The Big_Little architecture switches at coarse granularity and has separate core and memory systems, which cause more performance and power overhead on thread switch. The MRA scheme provides higher throughput/Watt than the Big_Little because of its ability to switch at fine granularity with reduced switching overhead. Thus, from Figure 5.4, we conclude that the NMRA_Fine scheme achieves higher throughput/Watt compared to the Fine_DVFS and MRA schemes. Figure 5.4 also compares the energy savings of several schemes. The largest energy savings of 34% are achieved using NMRA_Fine, followed by 24% using MRA, 19% using Big_Little and 17% using Fine_DVFS. The Fine_DVFS scheme achieve energy savings at the cost of performance loss. An average performance loss of 5.4% was observed when compared to running in the baseline architecture. The NMRA scheme with diverse core modes that resolve processor bottleneck, achieves a 7% average increase in performance compared to the baseline.

5.3.2 Power Efficiency Comparison between Oracular and PMC schemes

Figure 5.5 compares between the PMC and an Oracular scheme to determine core mode switch, across three different architectures that switch at fine granularity. In the oracular scheme, an oracle determines every 2K instruction, the best core mode for the next 2K instructions. The Oracular scheme provides an upper bound for the throughput/Watt that could be achieved but clearly, can not implemented in practice. The throughput/Watt achieved by the PMC-based scheme closely follows the oracular scheme in each of the different architectures.

5.3.3 Impact of switching overhead on different architectures

Figure 5.6 compares different architectures with varying mode switching overhead. Mode switching overhead is design dependent and hence it is important to estimate the impact on throughput/Watt of higher mode switching overheads. For

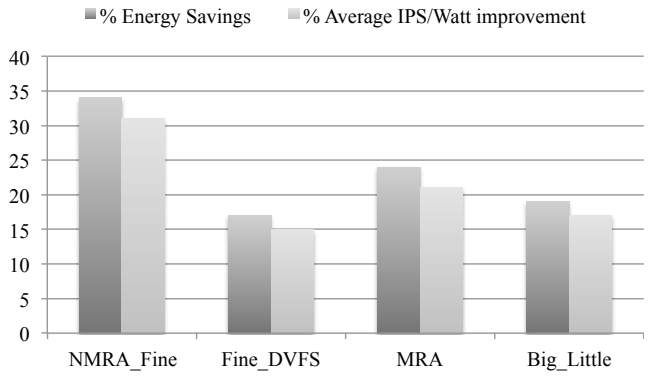

Figure 5.4. Throughput/Watt and energy savings comparison between DVFS, NMRA and BigLittle architectures.

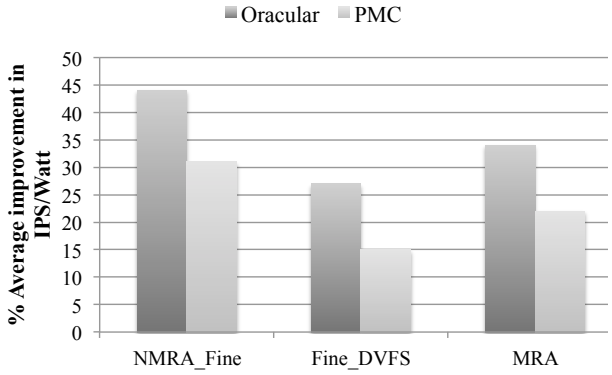

Figure 5.5. Throughput/Watt comparison across different architectures with PMC and oracular run time schemes.

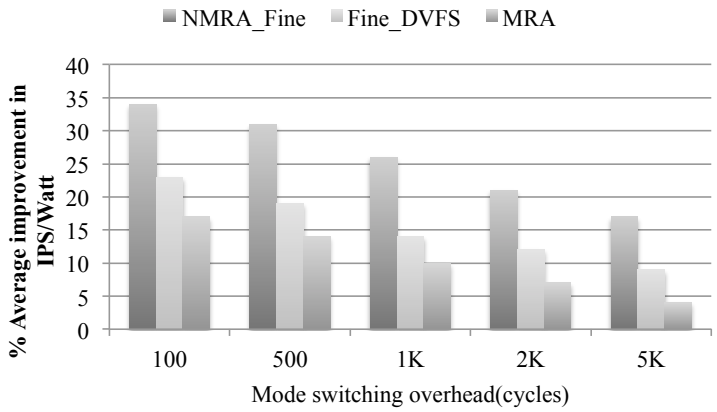

Figure 5.6. Impact of mode switching overhead on different architecture schemes.

the NMRA_Fine scheme, the average mode switching overhead was estimated to be 500 cycles, but the actual switching overhead is calculated in run time taking into account, time for draining the resources from the banked structures. The Fine_DVFS architecture incurs an overhead when switching between different V/F levels using an on-chip regulator and is estimated to be less than 200 cycles [29, 49]. For the MRA scheme, the overhead involves pipeline draining and clock gating the unused structures upon each mode switch and the average overhead is estimated to be less than 20nsec [61] or about 100 cycles [91]. From Figure 5.5, we observe that as the overhead increases from 500 cycles to 1K cycles, the throughput/Watt drops by 6%, 5% and 5.3% for the NMRA_Fine, Fine_DVFS and MRA schemes, respectively. A larger increase in the overhead beyond 1K cycles will result in a further loss in throughput/Watt indicating that architectures that switch at fine granularity need a fast switching mechanism.

5.4 Conclusion

Fine grain DVFS and reconfigurable architectures provide two alternatives for improving processor power efficiency. In this chapter, we compare the throughput/Watt and energy savings of various reconfigurable and DVFS architectures that support fine

grain and coarse grain switching. To evaluate the benefits of fine grain switching, two reconfigurable architecture were studied: non-monotonic reconfigurable architecture with four diverse core modes and reconfiguration between big and little cores. The use of an on-chip regulator that supports fine grain DVFS and the use of an off-chip regulator that can support only coarse grain DVFS was also studied. To evaluate power efficiency of various architecture, a PMC-based fast decision mechanism to support switching between various modes was implemented. Our results indicate that, when switching at fine granularity, the NMRA architecture provides 17% improvement in energy efficiency compared to fine grain DVFS alone and 11% more than MRA architecture that reconfigures between OOO and InO core. When compared to static Big/Little (Big_Little) architecture, NMRA architecture achieves 14% improvement in energy efficiency. Thus we conclude that at fine grain, non-monotonic core architecture with DVFS support is superior than either pure DVFS or Big/Little architecture.

CHAPTER 6

ON-LINE MECHANISM FOR RELIABILITY AND POWER-EFFICIENCY MANAGEMENT

In the previous chapters, we have explored dynamic AMP architecture for improved power efficiency. In this chapter, we follow the second approach and consider dynamic core reconfiguration from the perspectives of *both* throughput/Watt efficiency and vulnerability to soft-errors. A soft error can be caused by neutron or alpha particle strike which changes the state of a single bit as shown in Figure 6.1. The error produced by an alpha particle strike is transient and the bit is not permanently damaged.

Power efficiency and vulnerability to soft-error often lead to a trade-off in core reconfiguration. For example, a workload that exhibits frequent cache misses achieves a higher power efficiency under lower voltage and frequency conditions that lead to lower power without decrease in performance as the performance bottleneck is the result of cache misses and not low frequency. Even though this may increase power

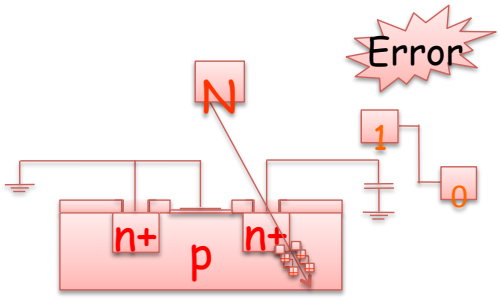

Figure 6.1. Soft error resulting in neutron or alpha particle strike resulting in a bit flip.

efficiency, it also leads to greater vulnerability to soft-error due to lower voltage. Recent literature has shown that reduced feature sizes and aggressive power management lead to increased soft error rate (SER) [13, 24]. Several studies report adverse impact of dynamic voltage and frequency scaling (DVFS) on SER [30, 88, 98, 106]. In this chapter, we investigate dynamic core reconfiguration with non-monotonic core types for optimizing two objectives simultaneously: improving throughput/Watt efficiency and reducing vulnerability to soft errors. Given that a thread is running on a certain core type, if by switching to a different core type the power efficiency can be improved without increasing the soft-error vulnerability, we should always do so. Similarly, if switching to another core configuration mode can reduce the soft-error vulnerability without decreasing the power efficiency, we should always do so. This approach ultimately leads to choices where we cannot improve one of these objectives without sacrificing the other one. We use a Cobb-Douglas production function [38] for arbitration among competing optimization objectives.

Measuring power efficiency and soft-error vulnerability requires quantitative metrics. For power efficiency we use the metric IPS/Watt, where IPS represents the number of instructions executed per second. Soft error vulnerability is measured in terms of Architecture Vulnerability Factor (AVF) introduced in [63]. AVF represents the fraction of fault that result in user visible error which is dependent on the processor micro-architecture as well as the application running on the processor. The AVF of a processor depends on the utilization of processor structures during runtime. Prior research has shown that AVF varies at instruction granularities within application [12]. AVF sensitivity to microarchitectural resource sizes was studied in [64]. These studies demonstrate that AVF varies from workload to workload and for a given workload, it varies from one core mode to another. Thus, in a dynamic core reconfiguration, there is a need to measure AVF online and take proactive actions,

which is a one of the contributions of this chapter. We summarize the work in this chapter below:

1. We develop a runtime mechanism for dynamically switching among non-monotonic core types, at fine grain granularity for simultaneously balancing throughput/Watt and SER.

2. We develop a runtime estimation mechanism for predicting throughput/Watt and AVF of all the core modes. The runtime mechanism estimates throughput/Watt and AVF in each of the core modes that are micro-architecturally different and run at different voltage/frequency using the performance monitoring counters (PMCs) of the host core mode.

3. We present a comparative study of the proposed runtime scheme against several other alternatives to demonstrate its efficiency.

6.1 Related Work

The probability that a soft error will lead to a user visible error is computed based on the processor AVF. The SER of a system is computed as the product of raw SER and individual component AVF [63]. Prior work have also shown how AVF may be estimated during runtime using a set of PMCs [63, 100]. Biswas et al. proposed quantized AVF estimation to track AVF variations at fine grain granularity using a few performance counters and also showed that the AVF of various processor components such as ROB, LSQ and IQ vary significantly during the course of program execution [12]. Soundararajan et al. studied the impact of frequency and voltage on SER when applying dynamic voltage and frequency (DVFS) [88]. Their work showed the impact of frequency and voltages changes on AVF and SER, when various DVFS algorithms are applied. They concluded that using only performance/Watt goals to choose various operating points for DVFS will not lead to improved system reliability.

Previous works have formulated online estimation models to compute AVF of a single core. In this work, we develop a linear regression based online model that can estimate AVF on all the core modes which are architecturally different and run at different voltage/frequency using the PMCs of the host core mode.

6.2 Proposed Architecture and Online Management

The different core modes used in this chapter for our experiments are similar to those shown in Table 4.2. To aid in fine grain switching between different modes, we only have one processor core whose resources are banked; they can be turned on or off and the frequency can be raised or lowered to configure the core to each of modes shown in Table 4.2.

We developed a run time management scheme that uses PMCs to select the core to reconfigure into. To balance between power and performance in choosing the right mode, we compute the metric IPS^2/Watt of all the modes . To compute IPS^2/Watt online, we use PMCs to estimate online, the power and performance for all the modes. Likewise, we use PMCs to predict AVF for all the modes, which would help us to compute the SER. The key novelty in this work, is that we need to predict IPS^2/Watt and AVF on all other modes, using the PMCs of the host mode unlilke previous works that compute AVF or power/IPC only on the same core. Thus, our online scheme will choose the best mode that balances throughput/Watt and SER.

6.2.1 AVF Estimation using PMCs

As the AVFs of ROB, LSQ and IQ vary significantly during the course of program execution, prior works have suggested using a small set of PMCs to track the AVF of processor components during run time [12]. We use PMCs to track AVF of the three components (ROB, LSQ and IQ) in each of modes using the PMCs of the host mode as in [12]. The counters used to track AVF include store buffer utilization,

Table 6.1. Accuracy of AVF estimation across all the modes

Core configurational Mode	R^2 coefficient
PMC AC ⇒ AVF	0.92
PMC NC ⇒ AVF	0.84
PMC LW ⇒ AVF	0.85
PMC SM ⇒ AVF	0.81

ROB utilization, branch miss-prediction, IQ utilization, ROB empty cycles and stores flushed before DTLB response. Counter values are collected every 2K instructions. Linear regression expressions were derived using these counters for estimating the overall AVF in each of the modes. The same set of training workloads are chosen as mentioned earlier for power/performance estimation to derive trained expressions to compute AVF online. Due to good correlation of the above mentioned counters with AVF, an average correlation coefficient R^2 of 0.86, considering all four modes, is obtained as shown in Table 6.2. Table 6.2 shows the average R^2 obtained when estimating AVF across different modes using the PMCs of the host mode. The accuracy in estimating the overall AVF across different modes is shown in Figure 6.2, for a mix of SPEC2000 and SPEC2006 benchmark [11]. We observe that the average error in estimating AVF is 11%, across all modes. Though the average error is 11%, the instantaneous error computed during run time may be much higher for some time intervals. The temporal distribution of the error is shown in Figure 6.3. For the sake of brevity, we show only the error distribution for modes which had the worst case average error, shown in Figure 6.2. As observed from Figure 6.3, for 75% of sample points the error is within ± 15% from the mean. This indicates that the overall error at the time of decision making is not high.

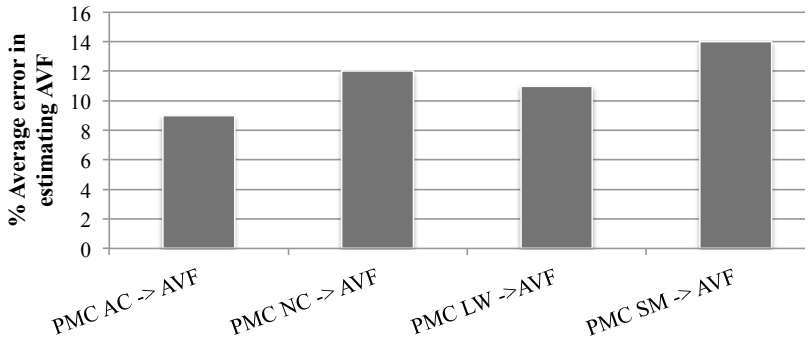

Figure 6.2. Average error in estimating overall AVF in all the modes using PMC of host core. For example, PMC AC ⇒ AVF denotes average error in estimating the AVF for all the other modes using the PMCs of AC mode.

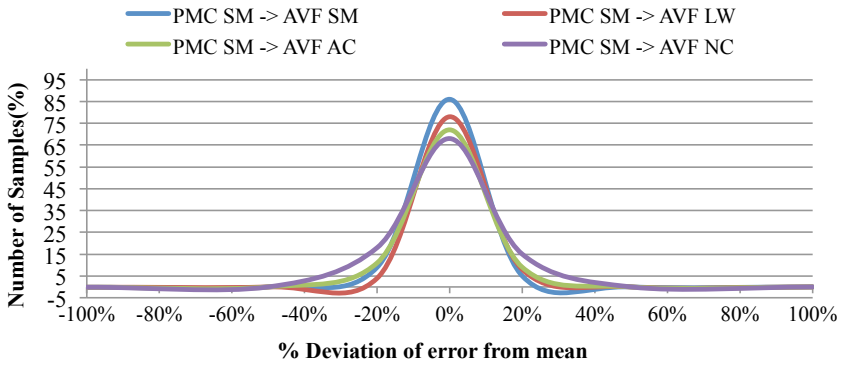

Figure 6.3. Distribution of error when using the PMCs of SM mode to compute the overall AVF for all other modes.

6.2.2 Metrics to achieve trade-off between throughput/Watt and reliability

Our on-line scheme consists of estimating the throughput/Watt and SER online using PMCs as discussed previously. IPS^2/Watt is computed online for all the modes using the PMCs of the current mode. We then compute the effective SER of each mode as shown below [63]:

Effective_SER = $AVF \times$ Raw_SER

Raw_SER is the total expected bit flip rate due to soft errors. As not all soft errors affect the output, *effective*_SER is derived from Raw_SER. The Raw_SER depends on the voltage and frequency. Scaling of voltage has an exponential relationship with soft errors, where a lower voltage leads to an increased error rate [31, 107]. Previous studies that have analyzed the effect of frequency on SER and have shown that SER has a linear relationship with frequency [31, 42]. The RAW_SER as a function of both voltage and frequency is shown below [31]:

Raw_SER(f,v) = $(f/f_{max}) \times e^{-c_0(v-v_{max})}$ * Raw_SER_0

f_{max} and v_{max} denote maximum voltage and frequency values for a core. c_0 is a constant as determined in [31]. Raw_SER_0 is the SER computed at max voltage and frequency. Decision to switch between the modes is made by computing a reliability power efficiency (RPE) metric online. The proposed RPE metric follows the Cobb-Douglas function [38] to trade off between two simultaneous objectives, namely, throughput/Watt and SER.

$$RPE = (\text{IPS}^2/\text{Watt})^a \times (\text{Effective\_SER})^{-b}$$

The exponents a and b are weights that control RPE, where a and b are numbers > 0 such that $a+b=1$. Normalization of IPS^2/Watt and Effective_SER is necessary, as RPE is a unitless metric. The values of the weights could be set based on the designers requirements as discussed in the result section.

For runtime management, the RPE metric is computed online for every mode in each of 2K instructions intervals. We call this scheme, PMC_RPE. A decision to switch configuration is based on maximizing RPE across modes. To prevent too frequent switching from one mode to another, the potential gain in RPE from switching core mode must exceed a certain threshold. The target threshold was found to be 4% based on a sensitivity study.

6.3 Experimental Setup and Results

We evaluate our proposed scheme using the Gem5 cycle accurate simulator [10] integrated with McPAT power model [59]. A modified Gem5 simulator was used to collect AVF information and power values online. SPEC2006 and SPEC2000 benchmarks are used for our experiments which were cross compiled for Alpha ISA with -O2 optimization. We ran our experiments for 4 billion instructions while skipping the first 2 billion.

6.3.1 Throughput/Watt and SER results

We evaluated the benefits of the proposed scheme using our 4-mode architecture and studied its impact on throughput/Watt and SER. Our baseline core is the average core, AC.

Benchmarks which are highly control bound (*astar, sjeng*) incur more stalls due to branch mispredictions. During periods of high misprediction, these benchmarks deliver the best throughput/Watt in the SM mode, as higher frequency or buffer resources do nothing to address this bottleneck. Similarly, memory intensive benchmarks (*mcf, soplex, libquantum*) contain low ILP phases due to significant data L2/TLB misses. These phases achieve a better throughput/Watt in NC or SM. As benchmarks like *mcf* have independent loads, NC with reduced resources sizes and increased frequency helps to improve performance. On the other hand, compute

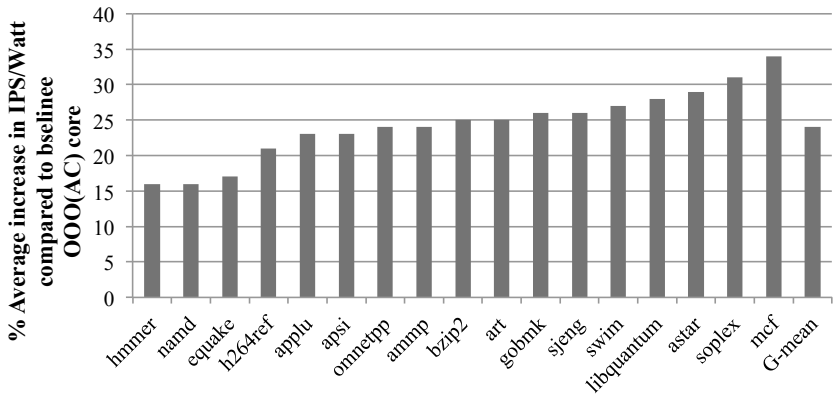

Figure 6.4. % Average increase in IPS/Watt compared to the baseline OOO(AC) using the PMC_RPE scheme

bound benchmarks (*bzip2, hmmer, h264ref*) have high IPC resulting in window bottleneck and thus prefer to have sections of application phases run on the LW as the performance of these benchmarks is limited by buffer resources and issue width.

Figure 6.4 shows improvement in IPS/Watt across all benchmarks using our PMC_RPE scheme. We observe that on average, an IPS/Watt benefit of 24% is achieved. The percentage decrease in Effective_SER compared to running in the baseline mode across all benchmarks is shown in Figure 6.5. Effective_SER reduction of 12% is achieved.

Figure 6.6 shows the time spent in each of the modes, showing that all modes are well utilized across all benchmarks. Memory intensive workloads such as *libquantum*, that have a high L2 miss rate and numerous parallel loads show high mode occupancy in SM and NM. When run on the baseline AC, *libquantum* has enough memory level parallelism to use the buffer resources but the amount of time instructions occupy entries in the ROB increases due to L2 misses. This would lead to a higher SER (resulting from increased AVF). If these sections of the program will be run on SM, it will provide a reduced SER (lower AVF) and a higher throughput/Watt compared to AC. Thus we observe that the percentage occupancy of SM mode is higher with RPE based scheme than with a scheme which only improves throughput/Watt as shown

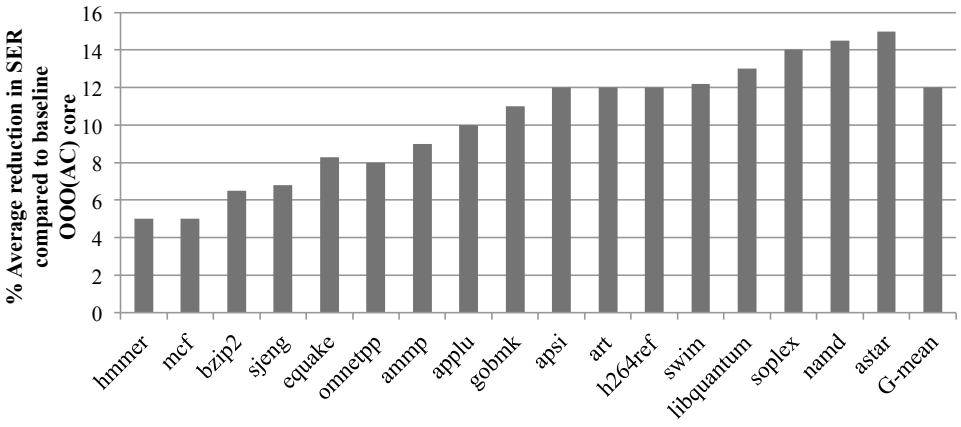

Figure 6.5. % Average decrease in Effective_SER compared to baseline OOO(AC) while switching based on the PMC_RPE scheme with RPE weight factors of a=0.6 and b=0.4.

in Figure 4.17. For *libquantum* we obtain 12% SER reduction and 26% improvement in throughput/Watt compared to the baseline core (AC) as shown in Figures 6.5 and 6.4, respectively.

Architecturally correct execution (*ACE*) bits are subset of processor state bits required for architecturally correct execution [63]. Some bits may not be critical to program execution and are termed as *unACE* bits. *unACE* bits include discarded bits due to mis-speculation. Only the ACE bits residence time in processor structures is taken into account for AVF and resultant SER computation. Memory intensive benchmarks like *mcf* show different behavior from *libquantum*. *mcf* experiences a significant number of branches (to be mispredicted or not) that depend on the long -latency load miss. If such a branch is mispredicted, all instructions in the ROB fetched after the branch instruction are un-ACE. Mis-predicted branch instructions are not part of ACE bits and thus do not affect AVF. Branch mispredictions that are independent of long-latency data cache misses will resolve quickly such that their interaction has a negligible effect on occupancy. Thus, running the benchmark on NC can provide higher throughput/Watt when compared to the baseline but the

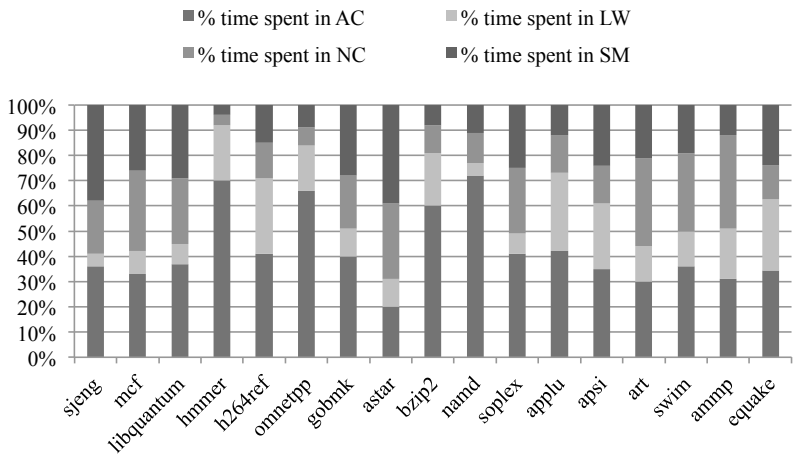

Figure 6.6. % of time spent in different modes

reduction in SER is not significant since occupancy of ACE bits is still limited by the miss-predicted branches. For *mcf* we obtain 5.3% reduction in SER and 34% improvement in throughput/Watt compared to the baseline as shown in Figures 6.5 and 6.4. Compute bound applications such as *hmmer* have high ILP and would prefer the LW mode. However, higher ILP in compute bound applications could cause the processor to bring more instructions into the pipeline, resulting in an increase in the number of ACE bits and thus increasing AVF. However, voltage and frequency scaling can have contrasting effect on Effective_SER as explained previously. Thus, switching into a new mode can increase AVF but might not increase the RAW_SER. The opposite situation can happen as well. For *hmmer* we obtain 5% reduction in SER and 16% improvement in throughput/Watt. Figure 6.7 shows the average (24%) improvement in IPS/Watt, performance increase (6%) and Effective_SER decrease (12%) compared to the AC baseline.

We also experimented with different weight factors for the RPE function to compare the improvement in throughput/Watt and average SER reduction for different weight factors. Figure 6.8 compares three different weight factors for RPE function. Setting the weight of IPS^2/Watt and Effective_SER to 0.7 and 0.3 respectively, results

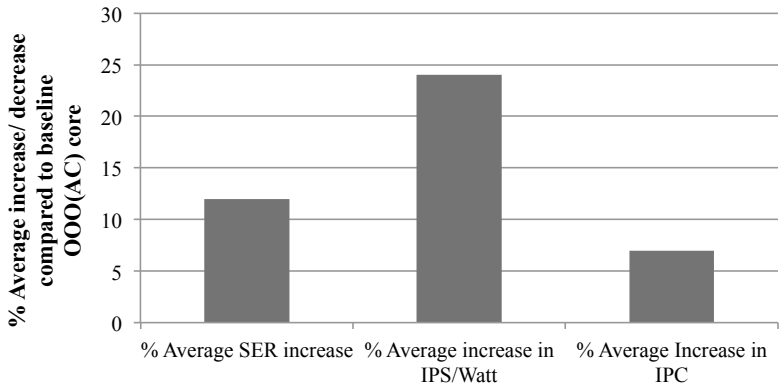

Figure 6.7. Increase in throughput/Watt, performance and decrease in Effective_SER while switching based on the PMC_RPE scheme compared to the baseline OOO(AC).

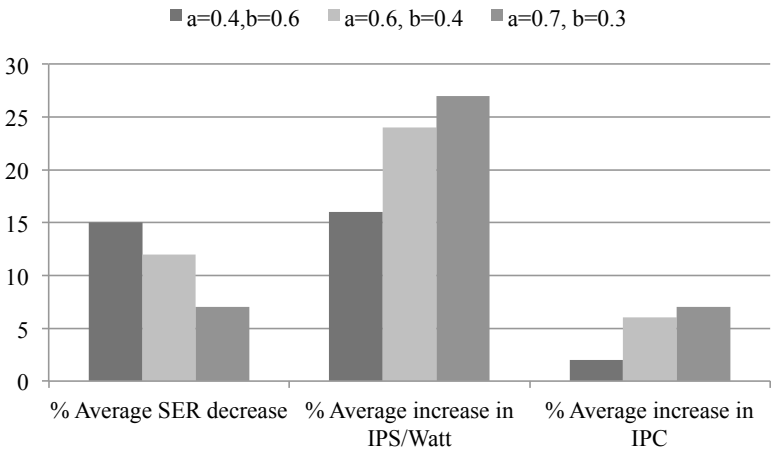

Figure 6.8. Variation in throughput/Watt, performance and Effective_SER while switching based on the PMC_RPE scheme.

in an increase in IPS/Watt by 28% and decrease in Effective_SER by 6% compared to AC baseline. Setting the weights to a=0.6 and b=0.4, results in 4% reduction in IPS/Watt and decrease in Effective_SER by 6% over the above mentioned weights (a=0.7, b=0.3). Further increase in weight of SER (a=0.4 and b=0.6), results in 3% more SER reduction and 8% less IPS/Watt compared to weights a=0.6 and b=0.4. Thus, based on the importance of particular metric for the designer, the weight factor for that metric can be increased or reduced.

6.3.2 Comparison to Alternative Switching Schemes

We compare our PMC_RPE scheme with three other schemes, namely, oracular scheme (Oracular), PMC-based coarse-grain switching scheme (CoarseGrain_RPE) and PMC based fine-grain switching scheme (PMC_IPS^2/Watt). Oracular scheme is implemented such that, every 2K instruction an oracle will determine what is the best mode to switch into for the next 2K instructions. The oracular scheme provides the upper-bound and can not be implemented in practice. As seen from Figure 6.9, the oracular scheme achieves a 11% higher average IPS/Watt and 4% higher average SER reduction compared to our fine grain (PMC_RPE) scheme. The CoarseGrain_RPE scheme makes switching decisions using the RPE metric at coarse grain granularity of 1M instructions. Due to coarse grain switching, this scheme does not take advantage of power benefits at fine granularity. AVF also varies at fine granularity which could affect the Effective_SER which this schemes fails to capture. CoareGrain_RPE achieves 8% lower IPS/Watt compared to PMC_RPE scheme. PMC_IPS^2/Watt scheme would try to switch into a mode that provides higher power efficiency without loosing much on performance. This scheme does not take SER into account while switching. Switching based on IPS^2/Watt provides 7% more IPS/Watt than the PMC_RPE scheme as its only goal is to improve throughput/Watt. We also obtain 6% reduction in Effective_SER compared to the PMC_RPE scheme.

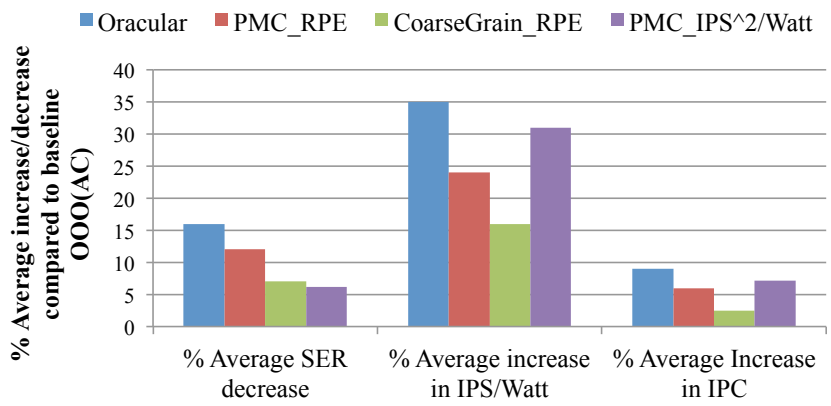

Figure 6.9. Comparison to various other switching schemes with RPE weight factors of a=0.6 and b=0.4.

Figure 6.10 shows the number of switches that take place in our 4 mode architecture with RPE based scheme at various instruction granularities. For our selected 2K instruction interval, we obtain 9200 switches per 100M instructions, i.e, the probability of making a mode switch is 18.2% every 2K instructions.

6.4 Conclusion

In this chapter we have presented an online management scheme for a reconfigurable architecture that strives to achieve a balance between power efficiency and reliability. The target reconfigurable architecture features four non-monotonic core configurations with varied micro-architectural resources, voltage and frequency. SER and throughput/Watt change rapidly during the course of program execution. The proposed runtime management scheme estimates on-line the power, performance and AVF based on a few performance counters to determine which is the best mode to run on for improving both power efficiency and AVF. Our results indicate that having diverse non-monotonic core types can increase the throughput/Watt of application by 24% while also providing a 12% reduction in SER compared to static execution on the baseline core.

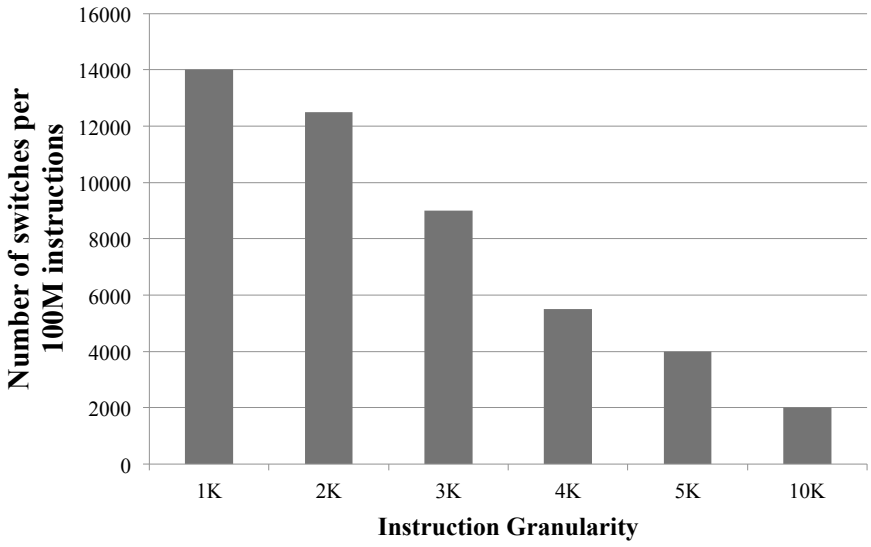

Figure 6.10. Number of switches per 100 million instructions for range of instruction granularities for our 4 mode morphing scheme.

CHAPTER 7

FUTURE DIRECTIONS

In this dissertation, we explored different kinds of core micro-architectures and online management schemes for improving core power efficiency. Future work based on this dissertation is presented next.

7.1 Co-Scheduling application between CPU and GPU

This dissertation focused on improving performance/Watt of applications using non-monotonic core types. The trend towards heterogeneous processors is continuing with tightly coupled accelerated processing unit (APU) designs in which the CPU and the GPU are integrated on the die and share on-die resources such as the memory hierarchy and interconnect. Challenging problem arises when we need to schedule applications between CPU consisting of non-monotonic core types and GPU. Both CPU and GPU may provide different performance and power profiles. The challenge is to design a run time applications scheduler that maps applications between CPU and GPU to achieve higher power efficiency.

7.2 Machine Learning based Online predictive model

We developed schemes to estimate performance and power on the host and other cores in the presence of DVFS. Potential future research is to find out whether we can further improve the accuracy of the estimation using machine learning based techniques such as neural network or support vector techniques.

BIBLIOGRAPHY

[1] Intel® core™ i7-900 processor ee, 32-nm process, datasheet, vol. 2.

[2] Albonesi, David H., Balasubramonian, Rajeev, Dropsho, Steven G., Dwarkadas, Sandhya, Friedman, Eby G., Huang, Michael C., Kursun, Volkan, Magklis, Grigorios, Scott, Michael L., Semeraro, Greg, Bose, Pradip, Buyuktosunoglu, Alper, Cook, Peter W., and Schuster, Stanley E. Dynamically tuning processor resources with adaptive processing. *Computer 36*, 12 (Dec. 2003), 49–58.

[3] Annamalai, A., Rodrigues, R., Koren, I., and Kundu, S. An opportunistic prediction-based thread scheduling to maximize throughput/watt in amps. In *Parallel Architectures and Compilation Techniques (PACT), 2013 22nd International Conference on* (2013), pp. 63–72.

[4] Annamalai, Arunachalam, Rodrigues, Rance, Koren, Israel, and Kundu, Sandip. Dynamic thread scheduling in asymmetric multicores to maximize performance-per-watt. In *Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2012 IEEE 26th International* (2012), pp. 964–967.

[5] Annavaram, M., Grochowski, E., and Shen, J. Mitigating amdahl's law through epi throttling. In *32nd International Symposium on Computer Architecture (ISCA'05)* (June 2005), pp. 298–309.

[6] Azizi, Omid, Mahesri, Aqeel, Lee, Benjamin C., Patel, Sanjay J., and Horowitz, Mark. Energy-performance tradeoffs in processor architecture and circuit design: A marginal cost analysis. In *Proceedings of the 37th Annual International Symposium on Computer Architecture* (New York, NY, USA, 2010), ISCA '10, ACM, pp. 26–36.

[7] Bahar, R.I., and Manne, S. Proceedings of the 28th annual international symposium on computer architecture. In *Power and energy reduction via pipeline balancing* (2001), pp. 218–229.

[8] Balasubramonian, Rajeev, Albonesi, David, Buyuktosunoglu, Alper, and Dwarkadas, Sandhya. Memory hierarchy reconfiguration for energy and performance in general-purpose processor architectures. In *Proceedings of the 33rd Annual ACM/IEEE International Symposium on Microarchitecture* (New York, NY, USA, 2000), MICRO 33, ACM, pp. 245–257.

[9] Becchi, Michela, and Crowley, Patrick. Dynamic thread assignment on heterogeneous multiprocessor architectures. In *Proceedings of the 3rd conference on Computing frontiers* (2006), CF '06, pp. 29–40.

[10] Binkert, Nathan, Beckmann, Bradford, Black, Gabriel, Reinhardt, Steven K., Saidi, Ali, Basu, Arkaprava, Hestness, Joel, Hower, Derek R., Krishna, Tushar, Sardashti, Somayeh, Sen, Rathijit, Sewell, Korey, Shoaib, Muhammad, Vaish, Nilay, Hill, Mark D., and Wood, David A. The gem5 simulator. In *ACM SIGARCH Computer Architecture News* (Aug 2011), vol. 39, pp. 1–7.

[11] Bird, Sarah, Phansalkar, Aashish, John, Lizy K, Mericas, Alex, and Indukuru, Rajeev. Performance characterization of spec cpu benchmarks on intel's core microarchitecture based processor. In *SPEC Benchmark Workshop* (January 2007), pp. 1–7.

[12] Biswas, Arijit, Soundararajan, Niranjan, Mukherjee, Shubhendu S, and Gurumurthi, Sudhanva. Quantized avf: A means of capturing vulnerability variations over small windows of time. In *IEEE Workshop on Silicon Errors in Logic-System Effects* (2009).

[13] Borkar, S., Karnik, T., and De, Vivek. Design and reliability challenges in nanometer technologies. In *Design Automation Conference, 2004. Proceedings. 41st* (july 2004), pp. 75–75.

[14] Borkar, Shekhar, and Chien, Andrew A. The future of microprocessors. In *Communications of the ACM* (2011), vol. 54.

[15] BorkarS. Designing reliable systems from unreliable components: the challenges of transistor variability and degradation. In *Micro, IEEE* (Nov. 2005), vol. 25, pp. 10–16.

[16] Branover, A., Foley, D., and Steinman, M. Amd fusion apu: Llano. *Micro, IEEE 32*, 2 (March 2012), 28–37.

[17] Burton, Edward, Schrom, Gerhard, Paillet, Fabrice, Douglas, James, Lambert, William J, Radhakrishnan, Krishnaja, Hill, Michael J, et al. Fivr—fully integrated voltage regulators on 4th generation intel® core™ socs. In *Applied Power Electronics Conference and Exposition (APEC), 2014 Twenty-Ninth Annual IEEE* (March 2014), pp. 432–449.

[18] Buyuktosunoglu, A., Karkhanis, T., Albonesi, D. H., and Bose, Pradip. Energy efficient co-adaptive instruction fetch and issue. In *Computer Architecture, 2003. Proceedings. 30th Annual International Symposium on* (june 2003), pp. 147–156.

[19] Chakraborty, Koushik, Wells, Philip M., and Sohi, Gurindar S. Computation spreading: Employing hardware migration to specialize cmp cores on-the-fly. In *Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems* (Dec 2006), vol. 34, pp. 283–292.

[20] Charles, James, Jassi, Preet, Ananth, Narayan S, Sadat, Abbas, and Fedorova, Alexandra. Evaluation of the intel core i7 turbo boost feature. In *Workload Characterization, 2009. IISWC 2009. IEEE International Symposium on* (oct. 2009), IEEE, pp. 188–197.

[21] Chen, Jian, and John, Lizy K. Efficient program scheduling for heterogeneous multi-core processors. In *Proceedings of the 46th Annual Design Automation Conference* (2009), DAC '09.

[22] Contreras, G., and Martonosi, M. Power prediction for Intel XScale reg; processors using performance monitoring unit events. In *Low Power Electronics and Design, 2005. ISLPED '05. Proceedings of the 2005 International Symposium on* (aug. 2005), pp. 221 – 226.

[23] Craeynest, K. Van, Jaleel, A., Eeckhout, L., Narvaez, P., and Emer, J. Scheduling heterogeneous multi-cores through performance impact estimation (pie). In *Proceedings of the 39th Annual International Symposium on Computer Architecture* (2012), ISCA '12, IEEE Computer Society, pp. 213–224.

[24] Dixit, A., and Wood, Alan. The impact of new technology on soft error rates. In *Reliability Physics Symposium (IRPS), 2011 IEEE International* (April 2011).

[25] Dolbeau, Romain, and Seznec, André. Cash: Revisiting hardware sharing in single-chip parallel processor. *Journal of Instruction-Level Parallelism 6*, 1–16 (2004).

[26] Dropsho, S., Buyuktosunoglu, A., Balasubramonian, R., Albonesi, D.H., Dwarkadas, S., Semeraro, G., Magklis, G., and Scottt, M.L. Integrating adaptive on-chip storage structures for reduced dynamic power. In *Parallel Architectures and Compilation Techniques, 2002. Proceedings. 2002 International Conference on* (2002), pp. 141–152.

[27] Dubach, Christophe, Jones, Timothy M., Bonilla, Edwin V., and O'Boyle, Michael F. P. A predictive model for dynamic microarchitectural adaptivity control. In *Proceedings of the 2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture* (Washington, DC, USA, 2010), MICRO '43, IEEE Computer Society, pp. 485–496.

[28] Esmaeilzadeh, Hadi, Blem, Emily, St. Amant, Renee, Sankaralingam, Karthikeyan, and Burger, Doug. Dark silicon and the end of multicore scaling. In *Computer Architecture (ISCA), 2011 38th Annual International Symposium on* (2011), IEEE, pp. 365–376.

[29] Eyerman, Stijn, and Eeckhout, Lieven. Fine-grained dvfs using on-chip regulators. *ACM Trans. Archit. Code Optim. 8*, 1 (Feb. 2011), 1:1–1:24.

[30] Firouzi, Farshad, Azarpeyvand, Ali, Salehi, Mostafa E, and Fakhraie, Sied Mehdi. Adaptive fault-tolerant dvfs with dynamic online avf prediction. In *Microelectronics Reliability* (2012), vol. 52, Elsevier, pp. 1197–1208.

[31] Firouzi, Farshad, Salehi, Mostafa E, Wang, Fan, and Fakhraie, Sied Mehdi. An accurate model for soft error rate estimation considering dynamic voltage and frequency scaling effects. In *Microelectronics Reliability* (2011).

[32] Fog, A. The microarchitecture of Intel, AMD and VIA CPU. Tech. rep., Copenhagen University College of Engineering.

[33] Goodacre, J. The homogeneity of architecture in a heterogeneous world. In *Embedded Computer Systems (SAMOS), 2012 International Conference on* (July 2012).

[34] Goulding-Hotta, Nathan, Sampson, Jack, Swanson, Steven, Taylor, Michael Bedford, Venkatesh, Ganesh, Garcia, Saturnino, Auricchio, Joe, Huang, Po-Chao, Arora, Manish, Nath, Siddhartha, et al. The greendroid mobile application processor: An architecture for silicon's dark future. In *Micro, IEEE* (March 2011), vol. 31, pp. 86–95.

[35] Greenhalgh, Peter. Big. little processing with arm cortex-a15 & cortex-a7. *ARM White Paper* (2011).

[36] Grochowski, E., Ronen, R., Shen, J., and Wang, P. Best of both latency and throughput. In *Computer Design: VLSI in Computers and Processors, ICCD 2004* (oct. 2004), pp. 236–243.

[37] Guthaus, M.R., Ringenberg, J.S., Ernst, D., Austin, T.M., Mudge, T., and Brown, R.B. Mibench: A free, commercially representative embedded benchmark suite. In *WWC-4* (2001).

[38] H.Douglas, Paul. Cobb, charles w and douglas, paul h. In *A Theory of Production* (1928).

[39] Held, Jim, Bautista, Jerry, and Koehl, Sean. From a few cores to many: A tera-scale computing research review. Tech. rep., Intel, 2006.

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

[41] Homayoun, H., Kontorinis, V., Shayan, A., Lin, Ta-Wei, and Tullsen, D.M. Dynamically heterogeneous cores through 3d resource pooling. In *High Performance Computer Architecture (HPCA), 2012 IEEE 18th International Symposium on* (Feb 2012).

[42] Irom, F., and Farmanesh, F.F. Frequency dependence of single-event upset in advanced commercial powerpc microprocessors. In *IEEE Transactions on Nuclear Science* (2004), vol. 51, pp. 3505–3509.

[43] Ishihara, T., and Yasuura, H. Voltage scheduling problem for dynamically variable voltage processors. In *Low Power Electronics and Design, 1998. Proceedings. 1998 International Symposium on* (Aug 2008), pp. 197–202.

[44] Keramidas, Georgios, Spiliopoulos, Vasileios, and Kaxiras, Stefanos. Interval-based models for run-time dvfs orchestration in superscalar processors. In *Proceedings of the 7th ACM International Conference on Computing Frontiers* (2010), pp. 287–296.

[45] Khan, Omer, and Kundu, Sandip. A self-adaptive scheduler for asymmetric multi-cores. In *Proceedings of the 20th symposium on Great lakes symposium on VLSI* (2010), ACM, pp. 397–400.

[46] Khan, Omer, and Kundu, Sandip. Empirical model for cooperative resizing of processor structures to exploit power-performance efficiency at runtime. *IET Circuits, Devices & Systems 6*, 5 (2012), 355–365.

[47] Khubaib, Suleman, M Aater, Hashemi, Milad, Wilkerson, Chris, and Patt, Yale N. Morphcore: An energy-efficient microarchitecture for high performance ilp and high throughput tlp. In *Microarchitecture (MICRO), 2012 45th Annual IEEE/ACM International Symposium on* (2012), IEEE, pp. 305–316.

[48] Kim, Changkyu, Sethumadhavan, Simha, Govindan, M. S., Ranganathan, Nitya, Gulati, Divya, Burger, Doug, and Keckler, Stephen W. Composable lightweight processors. In *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture* (2007), pp. 381–394.

[49] Kim, Wonyoung, Gupta, M. S., Wei, G. Y., and Brooks, D. System level analysis of fast, per-core dvfs using on-chip switching regulators. In *High Performance Computer Architecture, 2008. HPCA 2008. IEEE 14th International Symposium on* (feb. 2008), pp. 123–134.

[50] Kontorinis, Vasileios, Shayan, Amirali, Tullsen, Dean M., and Kumar, Rakesh. Reducing peak power with a table-driven adaptive processor core. In *Proceedings of the 42Nd Annual IEEE/ACM International Symposium on Microarchitecture* (2009).

[51] Koufaty, David, Reddy, Dheeraj, and Hahn, Scott. Bias scheduling in heterogeneous multi-core architectures. In *Proceedings of the 5th European conference on Computer systems* (2010), EuroSys '10, pp. 125–138.

[52] Kumar, Rakesh, Farkas, Keith I, Jouppi, Norman P, Ranganathan, Parthasarathy, and Tullsen, Dean M. Single-isa heterogeneous multi-core architectures: The potential for processor power reduction. In *Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture* (2003), IEEE, pp. 81–92.

[53] Kumar, Rakesh, Jouppi, Norman P, and Tullsen, Dean M. Conjoined-core chip multiprocessing. In *Proceedings of the 37th annual IEEE/ACM International Symposium on Microarchitecture* (2004), IEEE Computer Society, pp. 195–206.

[54] Kumar, Rakesh, Tullsen, Dean M, Jouppi, Norman P, and Ranganathan, Parthasarathy. Heterogeneous chip multiprocessors. *Computer 38*, 11 (2005), 32–38.

[55] Kumar, Rakesh, Tullsen, Dean M., Ranganathan, Parthasarathy, Jouppi, Norman P., and Farkas, Keith I. Single-isa heterogeneous multi-core architectures for multithreaded workload performance. *Proceedings of the 31st Annual International Symposium on Computer Architecture* (2004).

[56] Le, Hanh-Phuc, Crossley, J., Sanders, S.R., and Alon, E. A sub-ns response fully integrated battery-connected switched-capacitor voltage regulator delivering 0.19w/mm2 at 73 In *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2013 IEEE International* (Feb 2013), pp. 372–373.

[57] Le Sueur, Etienne, and Heiser, Gernot. Dynamic voltage and frequency scaling: The laws of diminishing returns. In *Proceedings of the 2010 International Conference on Power Aware Computing and Systems* (Berkeley, CA, USA, 2010), HotPower'10, USENIX Association, pp. 1–8.

[58] Lee, Chunho, et al. Mediabench: a tool for evaluating and synthesizing multimedia and communicatons systems. In *Proceedings of the 30th annual ACM/IEEE international symposium on Microarchitecture* (1997), MICRO 30.

[59] Li, S., Ahn, J. H., Strong, R. D., Brockman, J. B., Tullsen, D. M., and Jouppi, N. P. Mcpat: An integrated power, area, and timing modeling framework for multicore and manycore architectures. In *Proc. of the 42nd Annual International Symposium on Microarchitecture* (Dec 2009), pp. 469–480.

[60] Lukefahr, Andrew, Padmanabha, Shruti, Das, Reetuparna, Dreslinski, Jr., Ronald, Wenisch, Thomas F., and Mahlke, Scott. Heterogeneous microarchitectures trump voltage scaling for low-power cores. In *Proceedings of the 23rd International Conference on Parallel Architectures and Compilation* (New York, NY, USA, 2014), PACT '14, ACM, pp. 237–250.

[61] Lukefahr, Andrew, Padmanabha, Shruti, Das, Reetuparna, Sleiman, Faissal M, Dreslinski, Ronald, Wenisch, Thomas F, and Mahlke, Scott. Composite cores: Pushing heterogeneity into a core. In *Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture* (2012), IEEE Computer Society, pp. 317–328.

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

[63] Mukherjee, Shubhendu S., Weaver, Christopher, Emer, Joel, Reinhardt, Steven K., and Austin, Todd. A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor. In *Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture* (2003), pp. 29–.

[64] Nair, A.A., Eyerman, S., Eeckhout, L., and John, L.K. A first-order mechanistic model for architectural vulnerability factor. In *Computer Architecture (ISCA), 2012 39th Annual International Symposium on* (June 2012), pp. 273–284.

[65] Najaf-abadi, H.H., Choudhary, N.K., and Rotenberg, E. Core-selectability in chip multiprocessors. In *Parallel Architectures and Compilation Techniques, 2009. PACT '09. 18th International Conference on* (sep 2009), pp. 113–122.

[66] Najaf-abadi, H.H., and Rotenberg, E. Configurational workload characterization. In *Performance Analysis of Systems and software, 2008. ISPASS 2008. IEEE International Symposium on* (2008).

[67] Najaf-abadi, H.H., and Rotenberg, E. Architectural contesting. In *High Performance Computer Architecture, 2009. HPCA 2009. IEEE 15th International Symposium on* (Feb 2009).

[68] Navada, Sandeep, Choudhary, Niket K., and Rotenberg, Eric. Criticality-driven superscalar design space exploration. In *Proceedings of the 19th International Conference on Parallel Architectures and Compilation Techniques* (2010), pp. 261–272.

[69] Navada, Sandeep, Choudhary, Niket K, Wadhavkar, Salil V, and Rotenberg, Eric. A unified view of non-monotonic core selection and application steering in heterogeneous chip multiprocessors. In *Proceedings of the 22nd international conference on Parallel architectures and compilation techniques* (2013), IEEE Press, pp. 133–144.

[70] NVIDIA. Variable smp – a multi-core cpu architecture for low power and high performance. *Whitepaper* (2011).

[71] Padmanabha, Shruti, Lukefahr, Andrew, Das, Reetuparna, and Mahlke, Scott. Trace based phase prediction for tightly-coupled heterogeneous cores. In *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture* (2013), ACM, pp. 445–456.

[72] Park, Jaehyun, Shin, Donghwa, Chang, Naehyuck, and Pedram, M. Accurate modeling and calculation of delay and energy overheads of dynamic voltage scaling in modern high-performance microprocessors. In *Low-Power Electronics and Design (ISLPED), 2010 ACM/IEEE International Symposium on* (Aug 2010), pp. 419–424.

[73] Patsilaras, George, Choudhary, Niket K., and Tuck, James. Efficiently exploiting memory level parallelism on asymmetric coupled cores in the dark silicon era. In *ACM Trans. Archit. Code Optim.* (January 2012).

[74] Ponomarev, Dmitry, Kucuk, Gurhan, and Ghose, Kanad. Reducing power requirements of instruction scheduling through dynamic allocation of multiple datapath resources. In *Proceedings of the 34th Annual ACM/IEEE International Symposium on Microarchitecture* (2001), pp. 90–101.

[75] Pricopi, Mihai, and Mitra, Tulika. Bahurupi: A polymorphic heterogeneous multi-core architecture. *ACM Trans. Archit. Code Optim. 8*, 4 (Jan. 2012), 22:1–22:21.

[76] Raghavan, A., Emurian, L., Shao, Lei, Papaefthymiou, M., Pipe, K.P., Wenisch, T.F., and Martin, M.M.K. Micro, ieee. In *Utilizing Dark Silicon to Save Energy with Computational Sprinting* (Sept 2013), vol. 33, pp. 20–28.

[77] Rangan, Krishna K., Wei, Gu-Yeon, and Brooks, David. Thread motion: Fine-grained power management for multi-core systems. In *Proceedings of the 36th Annual International Symposium on Computer Architecture* (jun 2009).

[78] Rodrigues, R., Annamalai, A., Koren, I., and Kundu, S. A study on the use of performance counters to estimate power in microprocessors. In *IEEE Transactions on Circuits and Systems II: Express Briefs* (2013), vol. 60, p. 12.

[79] Rodrigues, R., Annamalai, A., Koren, I., Kundu, S., and Khan, O. Performance per watt benefits of dynamic core morphing in asymmetric multicores. In *Parallel Architectures and Compilation Techniques (PACT), 2011* (oct. 2011), pp. 121–130.

[80] Rodrigues, R., et al. Scalable thread scheduling in asymmetric multicores for power efficiency. In *Computer Architecture and High Performance Computing (SBAC-PAD), 2012 IEEE 24th International Symposium on* (oct. 2012), pp. 59 –66.

[81] Rodrigues, Rance, Annamalai, Arunachalam, Koren, Israel, and Kundu, Sandip. Improving performance per watt of asymmetric multi-core processors via online program phase classification and adaptive core morphing. *ACM Trans. Des. Autom. Electron. Syst. 18*, 1 (Jan. 2013), 5:1–5:23.

[82] Rusu, Stefan. Isscc 2013 trends. Tech. rep., International Solid-State Circuits Conference, feb. 2013.

[83] Saez, Juan Carlos, Prieto, Manuel, Fedorova, Alexandra, and Blagodurov, Sergey. A comprehensive scheduler for asymmetric multicore processors. In *In Proceedings of the 5th ACM European Conference on Computer Systems* (2010).

[84] Sawalha, L., Wolff, S., Tull, M.P., and Barnes, R.D. Phase-guided scheduling on single-isa heterogeneous multicore processors. In *Digital System Design (DSD), 2011 14th Euromicro Conference on* (2011), pp. 736–745.

[85] Shelepov, Daniel, Saez Alcaide, Juan Carlos, Jeffery, Stacey, Fedorova, Alexandra, Perez, Nestor, Huang, Zhi Feng, Blagodurov, Sergey, and Kumar, Viren. Hass: a scheduler for heterogeneous multicore systems. *SIGOPS Oper. Syst. Rev. 43*, 2 (April 2009), 66–75.

[86] Sherwood, Timothy, Sair, Suleyman, and Calder, Brad. Phase tracking and prediction. In *Proceedings of the 30th Annual International Symposium on Computer Architecture* (2003), pp. 336–349.

[87] Singh, Karan, Bhadauria, Major, and McKee, Sally A. Real time power estimation and thread scheduling via performance counters. *SIGARCH Comput. Archit. News 37*, 2 (July 2009), 46–55.

[88] Soundararajan, N., Vijaykrishnan, N., and Sivasubramaniam, A. Impact of dynamic voltage and frequency scaling on the architectural vulnerability of gals architectures. In *Low Power Electronics and Design (ISLPED), 2008 ACM/IEEE International Symposium on* (Aug 2008), pp. 351–356.

[89] SPEC2000. The Standard Performance Evaluation Corporation (Spec CPI2000 suite).

[90] Srinivasan, S., Kurella, N., Koren, I., and Kundu, S. Exploring heterogeneity within a core for improved power efficiency. In *IEEE Transactions on Parallel and Distributed Systems,* (2015), vol. 37, pp. 46–55.

[91] Srinivasan, S., Rodrigues, R., Annamalai, A., Koren, I., and Kundu, S. On dynamic polymorphing of a superscalar core for improving energy efficiency. In *Computer Design (ICCD), 2013 IEEE 31st International Conference on* (oct. 2013).

[92] Srinivasan, S., Rodrigues, R., Annamalai, A., Koren, I., and Kundu, S. A study on polymorphing superscalar processor dynamically to improve power efficiency. In *VLSI (ISVLSI), 2013 IEEE Computer Society Annual Symposium on* (Aug 2013).

[93] Srinivasan, Sadagopan, et al. Efficient interaction between OS and architecture in heterogeneous platforms. *SIGOPS Oper. Syst. Rev. 45* (February 2011), 62–72.

[94] Su, B., Gu, J., Shen, L., Huang, W., Greathouse, J. L., and Wang, Z. Ppep: Online performance, power, and energy prediction framework and dvfs space exploration. In *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture* (Dec 2014), pp. 445–457.

[95] Suleman, M. Aater, Mutlu, Onur, Qureshi, Moinuddin K., and Patt, Yale N. Accelerating critical section execution with asymmetric multi-core architectures. *SIGPLAN Not. 44*, 3 (Mar. 2009), 253–264.

[96] Tarjan, D., Boyer, M., and Skadron, K. Federation: Repurposing scalar cores for out-of-order instruction issue. In *Design Automation Conference, 2008. DAC 2008. 45th ACM/IEEE* (2008), pp. 772–775.

[97] Taylor, Michael B. Is dark silicon useful?: harnessing the four horsemen of the coming dark silicon apocalypse. In *Proceedings of the 49th Annual Design Automation Conference* (2012), ACM, pp. 1131–1136.

[98] Vadlamani, R., Zhao, Jia, Burleson, W., and Tessier, R. Multicore soft error rate stabilization using adaptive dual modular redundancy. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2010* (March 2010), pp. 27–32.

[99] Venkat, Ashish, and Tullsen, Dean M. Harnessing isa diversity: Design of a heterogeneous-isa chip multiprocessor. In *Proceeding of the 41st Annual International Symposium on Computer Architecuture* (jun 2014).

[100] Walcott, Kristen R., Humphreys, Greg, and Gurumurthi, Sudhanva. Dynamic prediction of architectural vulnerability from microarchitectural state. In *Proceedings of the 34th Annual International Symposium on Computer Architecture* (May 2007), pp. 516–527.

[101] Wang, Huaping, Koren, I., and Krishna, C.M. Utilization-based resource partitioning for power-performance efficiency in smt processors. In *Parallel and Distributed Systems, IEEE Transactions on* (july 2011), vol. 22, pp. 1150–1163.

[102] Winter, Jonathan A., et al. Scalable thread scheduling and global power management for heterogeneous many-core architectures. In *Proceedings of the 19th international conference on Parallel architectures and compilation techniques* (2010), PACT '10.

[103] Wunderlich, Roland E., Wenisch, Thomas F., Falsafi, Babak, and Hoe, James C. Smarts: Accelerating microarchitecture simulation via rigorous statistical sampling. In *Proceedings of the 30th Annual International Symposium on Computer Architecture* (may 2003).

[104] Xie, Fen, Martonosi, M., and Malik, S. Bounds on power savings using runtime dynamic voltage scaling: an exact algorithm and a linear-time heuristic approximation. In *Low Power Electronics and Design, 2005. ISLPED '05. Proceedings of the 2005 International Symposium on* (Aug 2005).

[105] Xu, Bingxiong, and Albonesi, David H. A methodology for the analysis of dynamic application parallelism and its application to reconfigurable computing. In *Proc. SPIE Int'l Symp. Reconfigurable Technology: FPGAs for Computing and Applications* (1999), pp. 78–86.

[106] Xu, Xin, Teramoto, K., Morales, A., and Huang, H.H. Dual: Reliability-aware power management in data centers. In *Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on* (May 2013), pp. 530–537.

[107] Zhu, D., Melhem, R., and Mossé, D. The effects of energy management on reliability in real-time embedded systems. In *ICCAD* (2004), pp. 35–40.