November 2015

ADACORE: Achieving Energy Efficiency via Adaptive Core Morphing at Runtime

Nithesh Kurella
University of Massachusetts Amherst

ADACORE: ACHIEVING ENERGY EFFICIENCY VIA ADAPTIVE CORE MORPHING AT RUNTIME

A Thesis Presented

by

NITHESH KURELLA

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

MASTER OF SCIENCE

September 2015

Electrical and Computer Engineering

ADACORE: ACHIEVING ENERGY EFFICIENCY VIA ADAPTIVE CORE MORPHING AT RUNTIME

A Thesis Presented

by

NITHESH KURELLA

Approved as to style and content by:

Israel Koren, Co-chair

Sandip Kundu, Co-chair

Wayne Burleson, Member

C. V Hollot, Department Chair
Electrical and Computer Engineering

ACKNOWLEDGMENTS

This thesis would not be complete without acknowledging all the wonderful people who helped and inspired me during my time in graduate school.

I am deeply indebted to my advisors Professor Sandip Kundu and Professor Israel Koren for their guidance and support. Their vision, enthusiasm and attention to detail would always continue to inspire and motivate me. I thank Professor Wayne Burleson for his valuable time and advice initially as my academic advisor and later as a member of my thesis committee.

I am grateful to my colleague Sudarshan Srinivasan for his help and insights throughout this exciting effort. I would like to thank Rance Rodrigues for his mentorship and lab mates Ye, Arun, Shikang and Mayank for all the fun times in KEB-310. I would like to express my gratitude to my friend Divyashree Bhat whose encouragement gave me the confidence to pursue this thesis.

Lastly, I would like to thank my family, friends and god for all the love and support through the years.

ABSTRACT

ADACORE: ACHIEVING ENERGY EFFICIENCY VIA ADAPTIVE CORE MORPHING AT RUNTIME

SEPTEMBER 2015

NITHESH KURELLA

B. E., VISVESVARAYA TECHNOLOGICAL UNIVERSITY

M. S., UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Professor Israel Koren and Professor Sandip Kundu

Heterogeneous multicore processors offer an energy-efficient alternative to homogeneous multicores. Typically, heterogeneous multi-core refers to a system with more than one core where all the cores use a single ISA but differ in one or more micro-architectural configurations. A carefully designed multicore system consists of cores of diverse power and performance profiles. During execution, an application is run on a core that offers the best trade-off between performance and energy-efficiency. Since the resource needs of an application may vary with time, so does the optimal core choice. Moving a thread from one core to another involves transferring the entire processor state and cache warm-up. Frequent migration leads to large performance overhead, negating any benefits of migration. Infrequent migration on the other hand leads to missed opportunities. Thus, reducing overhead of migration is integral to harnessing benefits of heterogeneous multicores.

This work proposes *AdaCore*, a novel core architecture which pushes the heterogeneity exploited in the heterogeneous multicore into a single core. *AdaCore* primarily

addresses the resource bottlenecks in workloads. The design attempts to adaptively match the resource demands by reconfiguring on-chip resources at a fine-grain granularity. The adaptive core morphing allows core configurations with diverse power and performance profiles within a single core by adaptive voltage, frequency and resource reconfiguration. Towards this end, the proposed novel architecture while providing energy savings, improves performance with a low overhead in-core reconfiguration. This thesis further compares *AdaCore* with a standard Out-of-Order core with capability to perform Dynamic Voltage and Frequency Scaling (DVFS) designed to achieve energy efficiency.

The results presented in this thesis indicate that the proposed scheme can improve the performance/Watt of application, on average, by 32% over a static out-of-order core and by 14% over DVFS. The proposed scheme improves $IPS^2/Watt$ by 38% over static out-of-order core.

TABLE OF CONTENTS

LIST OF TABLES

Table **Page**

LIST OF FIGURES

Figure **Page**

CHAPTER 1

INTRODUCTION

1.1 Objective

Microprocessor design over the last few decades leveraged semiconductor industry's ability to pack more transistors on chip to extract higher performance. The higher performance obtained is due to the extraction of instruction level parallelism (ILP) in a superscalar Out Of Order (OOO) processors. The limited availability of ILP coupled with quest for efficient use of available resources led researchers to explore multiple processors on the same chip. Thread level parallelism was exploited in applications by dividing the workload execution amongst multiple copies of the same processor [29]. However, in recent years the need for energy efficient high performance computing has continuously increased. In data center space, the energy efficiency of processors directly translate to operating costs. Meanwhile, in the mobile phone space, energy efficient processors lead to improved battery life. Correspondingly, as the transistor feature size has reduced to nanometer regime, the operating voltage did not reduce proportionally [13]. These factors therefore have called for a renewed search for energy efficiency in processor design.

Heterogeneous multi-core processors (HMPs) have been proposed as an alternative to regular multi-cores to improve energy efficiency. HMPs comprise of cores that implement the same ISA but differ in performance and energy characteristics due to varying sizes of micro-architectural resources. Previous research on HMPs have identified multiple opportunities for exploiting heterogeneity in the multi-core paradigm. Kumar *et al.* proposed using a mix of cores with different power and performance

characteristics, so that a program phase of an application is mapped to a core which achieves the best energy-efficiency [22]. For example, if a program phase exhibits low Instructions Per Cycle (IPC), it can be mapped to a small core for greater energy efficiency. In such a case, boosting the frequency of the smaller core (within power dissipation constraints) may achieve *both* higher performance and higher energy efficiency. In another use case, the serial sections of a multi-threaded program may be sped up [2] by using a big core, while parallel executions are run on moderately resourced cores to improve overall performance within a power budget. The focus of this research is on the former use-case, where the goal is to design an architecture that improves energy efficiency of a single thread without significant sacrifice to its performance.

Some HMP design references feature a high performance OOO core combined with a smaller low performance and energy efficient In-Order (InO) core. ARM has designed such an AMP which they call big.Little™with the above mentioned core types [11]. In such an HMP, compute intensive phases of an application are run on a high performance core while low performance phases such as memory intensive phases are executed on an InO core resulting in improved energy efficiency. However, a workload's performance on a particular core type depends on larger set of processor features since workloads are diverse in nature. Workloads could be highly memory intensive or may exhibit high dependency between instructions. Some workloads may experience higher rates of branch mis-prediction, while some exhibit large exploitable instruction level parallelism. Often a workload might stall due to lack of adequate resources in load-store queue (LSQ), re-order buffer (ROB), issue queue (IQ) or execution resources. Thus, a diverse set of cores are needed that could address varying resource needs at various program phases of an application.

Kumar *et al.* considered HMPs appropriate for various program phases [23]. Navada *et al.* explored non-monotonic HMP cores which are specifically designed

to address bottlenecks that result in poor performance [28]. Common performance bottleneck in cores arise from cache misses, limited execution resources or execution width, large amounts of instruction dependencies, or inherently low instruction level parallelism. Thus, if the designed core types explicitly address such bottlenecks and with effective runtime mechanisms, performance and energy efficiency of a workload can be improved. Accordingly, Navada *et al* proposed a set of distinct core types consisting of narrow core which suits applications with low ILP, a large window core for application phases that have window bottlenecks (reorder buffer, issue queue), and a wider core for phases that have width bottlenecks (fetch and issue width), and an average super-scalar OOO core to target most common scenarios [28].

Assigning the application to the most efficient core in an HMP is usually accomplished by a process termed as *sampling*. The application is briefly run on each of the available cores and then assigned to the core that best suits the performance needs of the workload [22]. Migrating an application from one core to another involves significant overhead as the complete state of the application must be transferred to the new core. It is to be noted that in sampling, state has to be transferred as many times as the number of additional cores. To reduce the impact of this overhead, thread migration in HMP is usually done at a coarse grain instruction granularity, typically measured in tens to hundred of millions of instructions [16]. This approach works well when the program phases last for millions of instructions.

Recent research has shown that energy saving opportunities can often be found in much shorter time scales, typically measured in thousands of instructions. If these relatively short phases could be mapped to the most energy efficient core, the overall energy efficiency would improve. To support such fine grain switching, Lukefahr *et al* proposed a morphable architecture where an OOO core dynamically morphs into an InO core at runtime [17, 27]. In these fine grain morphable core designs, the workload continues to execute on the same core, and as a result, the overhead associated with

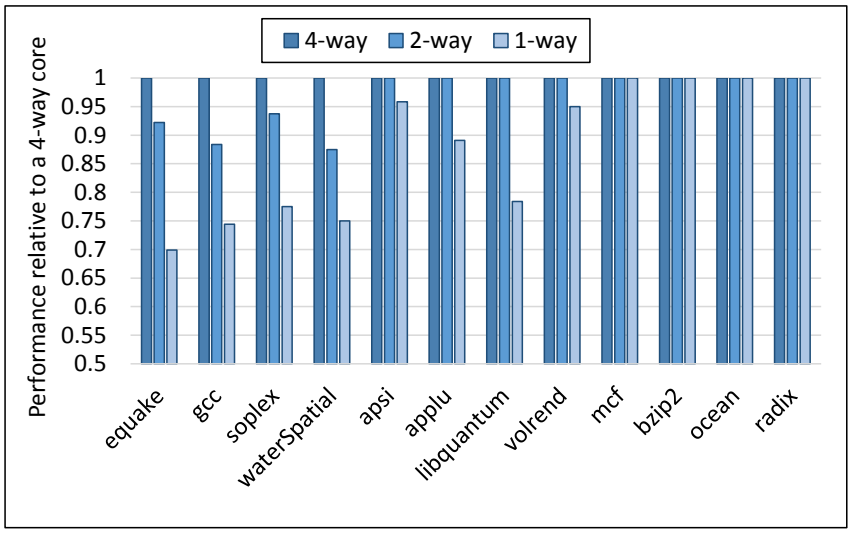

Figure 1.1: Performance of a 4-way, 2-way and 1-way OOO core relative to the 4-way core for various workloads from the SPEC 2000, SPEC 2006 and SPLASH-2 suites.

transferring the entire state of the workload to another core is eliminated. This allows morphing to take place at finer instruction granularities (∼1000 instructions) which reportedly results in significant energy savings at a small loss in performance [17, 27]. However, this approach has the following shortcomings:

1. Core Alternatives: Morphing was considered only between two extreme architectures, i.e., a wide issue OOO core and an InO core. However, as discussed previously, the resource bottlenecks or excesses can be quite diverse, and as a result important power and performance optimization opportunities, may be missed.

   To study the potential benefits of having more core types, we experimented with three OOO cores with varying execution widths, frequencies and resources scaled appropriately for each width. We call these cores the 4-way, 2-way and 1-way cores, indicating the respective execution widths. In Figure 1.1 we present the performance of these three options, relative to that of the 4-way core, for workloads from the SPEC 2000 [39], SPEC 2006 [5] and SPLASH-2 [44] suites.

It can be seen that there are some workloads that would benefit from the 4-way core but there are other workloads for which a 2-way or 1-way core provides adequate performance. The 2-way and 1-way cores do not need the same level of resources as the 4-way core. Hence, for such workloads there is a potential for power savings by running them on the reduced fetch width core, resulting in better performance-per-power. This shows that a collection of intermediate architectures may provide considerable benefits.

2. Hardware Complexity: The complexity of previously proposed morphing schemes is high. Lukefahr *et al* [27] proposed incorporating two execution backends in the same core. Khubaib *et al* [17] proposed converting an OOO into a highly threaded in-order simultaneous multi-threaded (SMT) execution. Both designs entail substantial changes to the micro-architecture to make a super-scalar OOO core to turn into an InO core.

Navada *et al.* [28] have performed a complete core design exploration and came up with a set of non-monotonic core types that can improve the performance/power of a single threaded processor. We rely on these core design configurations and incorporate all of them in a single core to devise a morphable architecture.

This thesis presents *AdaCore*, a novel morphable architecture in which the core adaptively morphs by switching between several OOO core modes where each mode may differ from the other in fetch and issue width, buffer sizes (IQ, LSQ, ROB) and clock frequency. We term this morphable architecture as *AdaCore* considering the architecture is constantly adapting. *AdaCore* can seamlessly switch between one core-mode to another at a fine grain instruction granularity. This simplifies the implementation as the transition is from one OOO core-mode to another and does not involve significant micro-architecture changes.

This thesis also attempts to investigate whether other energy efficient design techniques such as scaling processor voltage and frequency dynamically at a fine grain

granularity can achieve similar energy efficiency as *AdaCore*. For example, application phases which are memory bound or phases with large branch mis-predictions may not gain much in performance with a higher frequency. In such scenarios, Dynamic Voltage and Frequency Scaling (DVFS) can potentially improve energy efficiency. Furthermore, the ability to scale the voltages and frequencies at fine grain granularity may provide an improved chance of tending to short-lived phases thereby extracting energy efficiency at a fine grain. To this end, we design a runtime scheme to scale voltage and frequency on a standard OOO core and compare the energy efficiency achieved by such a design to that of *AdaCore*.

1.2 Contributions

The major contributions of this work include:

- A morphable core architecture that is capable of switching between various OOO core-modes. The mode switching is fast, allowing us to address application's varying resource demands at a fine grain granularity while mapping the application to the most energy efficient core.

- A simple online mechanism that can estimate the application performance and power across all available core modes. The decision to switch can be made at *fine grain instruction granularity* that can not be accomplished using conventional sampling based techniques.

- An energy efficiency comparison of the morphable core to an OOO superscalar core with DVFS.

CHAPTER 2

PRIOR RESEARCH

In this chapter, a brief summary of the relevant prior research related to asymmetric multi-core processors, morphable core architectures, application to core assignment and dynamic voltage and frequency scaling is presented.

2.1 Dynamic Voltage and Frequency Scaling

DVFS is a technique traditionally used in processors with single and multiple cores. It is used as a means to keep the processor running at maximum possible frequency as long as it is possible. The moment a processor hits a thermal threshold, the voltage and in turn frequency of the processor are reduced to protect the processor from crossing the power dissipation limits [26]. Such throttling of frequency and voltage at runtime due to thermal emergencies is one of the important applications of DVFS.

The performance of certain sections of a workload can be improved by increasing the frequency of the processor. To increase the frequency, the voltage may also have to be increased. Intel implements a similar DVFS scheme to boost the frequency of the processor as high as 3.2GHz to speed up parts of the application for a short period of time [7].

Another application of DVFS is to achieve energy efficiency by scaling the voltage and frequency down at the expense of performance [36]. Researchers have proposed several runtime procedures the processor can follow in order to perform DVFS. The proposals range from the traditional method of relying on the operating system to perform DVFS, to using the compiler to predict the need for and trigger DVFS [45].

Several schemes include DVFS decision based on hardware performance counters [1],[41]. Annamalai *et al.* use a regression analysis based approach to use performance counters to estimate performance and power [1]. The estimations are in turn used to make the decision for triggering DVFS. The voltage and frequency scaling is performed within a asymmetric multi-core system. They conclude that a combination of thread swapping in asymmetric multi-cores and DVFS results in improved energy efficiency.

2.2 Heterogeneous Multicore Processor (HMP) Designs

Heterogeneous multi-core systems include at least one core that differs from the rest in its configuration. Such a system may help multi-threaded, multi-programmed and even a single threaded applications to achieve energy efficiency or gain performance or both. To improve the performance of a multi-threaded application within a power dissipation limit, researchers have proposed HMP designs which include a single large OOO processor and multiple moderately resourced OOO processors [2]. Similarly, authors in [42] use a large OOO core to improve performance during serial portions of a multi-threaded application. The parallel sections of the same workload were assigned to several smaller cores to make the execution of the light-weight threads energy efficient.

A single threaded workload may exhibit several phases during the course of its execution. To ensure that each of the application phases is executed in the most energy efficient core, Kumar *et al.* proposed an HMP design [22] consisting of cores with diverse power and performance characteristics. Each application program phase is executed on the most energy efficient core [22]. For example, if an application is experiencing low performance on a large OOO core, the frequency of the processor can be increased to improve performance. However, it may not be the most energy efficient. Such phases, if executed on a small core, may achieve higher energy efficiency. The authors observed that scaling the frequency of the smaller core (within power

budget) may achieve *both* higher performance and higher energy efficiency. Kumar *et al.* extended the proposal to multi-programmed workloads and designed a HMP system which can improve performance [24].

Grochowski *et al.* [12] propose a mix of asymmetric cores coupled with voltage and frequency scaling to consume the least amount of energy per instruction. Navada *et al.* in [28] consider accelerating single threaded workload by designing a set of cores that solve various application performance bottlenecks. They performed a complete design space exploration and identified a set of heterogeneous cores that would maximize performance. Their research suggested that with N core types, the optimal set 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.

The above described HMP architectures make thread scheduling decisions at coarse grain instruction granularity or granularity of phase change in application (about 100 million instructions). An attempt to make the thread to core mapping at a finer granularity incurs a larger thread swapping penalty in terms of performance and power.

2.3 Morphable Core Architectures

There have been few proposals advocating dynamic reconfiguration of cores during runtime. Such reconfigurations enable cores to adapt according to the workload and improve performance and/or power efficiency. Several other publications consider a multi-core system in which a number of small cores fuse together to form a large OOO core on demand [18, 33]. These approaches introduce additional latencies in the pipeline due to the combining of resources from various cores.

Recently, morphable architectures have been proposed to support fine grain configuration switching and improve performance/Watt. In the architecture proposed

by Khubaib *et al.* in [17], a traditional OOO core is morphed into an in-order SMT core when the application enters the parallel sections of the workload. The authors observe that a multi-threaded in-order core can achieve higher energy efficiency than a superscalar OOO core for certain multi-threaded application phases. However, their design requires significantly complex hardware changes to the micro-architecture to construct an in-order core (that can support SMT) from an OOO core.

Lukefahr *et al.* [27] proposed a morphable architecture where an OOO core can morph into an InO by switching between two execution back-ends in the same core but inlcude a common front-end. One of the back-end engines is used in the OOO mode while the other in the InO mode. They term this architecture as a *Composite Core*. With their composite core architecture, the core can switch between OOO and InO modes at fine grain instruction slices. As a result, even short low performance phases are executed in the energy efficient InO mode. The approach therefore, provides the application an option to search for improved performance or energy efficiency between the two modes which have highly different characteristics. However, the two modes may not cater to the demands of all the single-threaded application phases.

2.4 Application Scheduling

Scheduling the application threads to the most energy or performance efficient cores or core modes in HMP or morphable cores, respectively, require an efficient scheduling mechanism. Several researchers utilize offline workload analysis to understand application behavior. Observations from offline methods such as regression-based analysis are used to schedule threads in heterogeneous multi-core systems [16, 37]. The scheduling based on such offline analysis involves studying the workload behavior offline. The knowledge obtained prior to the application execution is applied at runtime to achieve optimal scheduling. However, offline analysis schemes are not

always the most practical. This is due to their inability to schedule new applications whose behavior may not have been analysed previously.

A better and viable solution to the application scheduling problem would be an online learning mechanism. Such a scheme can learn the characteristics of the applications during their execution. It can use the obtained behavioral observations to make an informed thread scheduling decision. An online learning scheme may rely on phase classification to identify the application phase coupled with sampling techniques to identify the best schedule [3, 22, 35, 43]. Whenever the mechanism identifies a change in the application phase, the application is sampled on all the available cores in the HMP. The efficiency of each of the cores during the sampled period is observed and the application is assigned to the best core. Such a mechanism which samples application on multiple cores to decide on the best core, causes a large performance penalty. Therefore, sampling based schemes can not be applied at fine grain instruction granularities.

Ability to perform thread scheduling at a fine grain granularities can provide a better opportunity to execute the application on the most efficient core or core-mode. To this end, estimation-based runtime scheduling mechanisms were proposed by researchers. In such schemes, the performance and/or power of an application phase on each of the available configurations is estimated in runtime. The estimation mechanism uses statistics such as cache misses and pipeline stalls recorded on the current core configuration to extrapolate the performance and power on the other configurations [8, 21, 27, 34, 40]. Since the performance estimations can be performed at any desired granularity, estimation-based online schemes outperform profiling/sampling-based online schemes described above.

CHAPTER 3

PROPOSED ARCHITECTURE

For the reasons explained in Chapter 2, the efforts of this thesis have been focussed on designing a core architecture with capability to dynamically match the application's resource demands. A core with such a capability would belong to the morphable core category of core design described in Chapter 2. Before we venture into the reconfigurablity of the proposed core, we first need to determine the modes of execution that should be present in the *AdaCore*.

3.1 Selection of Core-Modes

Heterogeneous multi-core architecture typically contains cores of different types such that each core-type can satisfy a specific application phase resource demands. Each phase in an application has a certain degree of ILP. The thermal dissipation of a package (TDP) limits the overall power dissipation. Thus, a core cannot feature the largest possible microarchitectural resources for all structures, yet operate at the highest possible frequency. To support a core with a high ILP, the size of the small on-chip buffers that extract the ILP should be increased or the width of the pipeline stages should be resized. These changes might require additional circuits which in turn, will affect core frequency due to the TDP limit. Thus, designing the right mix of cores for HMP that caters to demands of all application phases requires careful balancing.

Navada *et al.* chose core-types for a heterogeneous multi-core design similar to a scenario described above. They do so by performing in-depth RTL based design space

exploration using a genetic algorithm based approach [28]. Core parameters such as fetch width, issue width, load store queue, physical register file, L1 I and D caches and clock frequency were extensively explored by them resulting in 13,966 design points. Their key finding was that if there is only one core-type, then the best core type (named the average core) resembles an existing commercial super-scalar OOO core, where the core parameters are chosen to strike a balance between achieving sufficient ILP and frequency. The alternate cores in an HMP would be designed to relieve bottlenecks that arise in the average core. Their choice of core design parameters obtained from design space exploration is summarized in Table 3.1 which shows a set of non-monotonic core-types with unconstrained power.

We observe from Table 3.1 that, along with the average OOO (AC) core, the design choices include a narrow core (NC) that has smaller fetch/issue width and lower frequency, a larger window core (LW) that has increased window sizes, and a wider width core (WC) that relieves width bottleneck. In their scheme, Navada *et al.* switch a program from one core to another during the course of its execution to improve performance and energy efficiency at a coarse grain granularity. Since the cache contents cannot be migrated easily, core hopping cannot be performed frequently [28]. We observe further that the featured wider core in [28] has a bigger I-Cache to overcome I-cache bottleneck. However, it has been reported in [15] that even with cache size of 32KB for L1, instruction misses are rather infrequent, except for a few SPEC 2006 benchmarks. Consequently, we have chosen I-cache and D-cache sizes of 64KB.

In this work, the interest lies in designing a morphable core architecture that would enable the core to cater to application phase demands at a fine grain granularity. Therefore, the proposed *AdaCore* consists of all the core-types proposed by Navada *et al.* as core-modes in a single reconfigurable core.

Table 3.1: Micro-architecture parameters of selected core-modes.

Mode	ClockPeriod (ns)	Buffer size (IQ,LSQ,ROB)	Width (fetch,issue)
AC	0.6	32,128,128	3,4
NC	0.5	32,64,64	2,2
LW	0.7	48,128,384	4,4
WC	0.7	32,128,128	6,6

3.2 Adaptive Core Morphing

In the proposed scheme, 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 any of the available core-modes shown in Table 3.1.

The baseline core-mode is the average OOO core that would adaptively morph into three other core-modes, namely, wider core-mode, narrow core-mode or larger window core-mode at runtime. The four distinct core-modes have different buffer sizes, fetch and issue width and also run at different frequencies. However, they all have the same cache size. This will allow the resources to be resized dynamically leaving the content of the cache intact which in turn enables fast switching between core-modes creating opportunities to explore frequent switching between modes.

Fine grain switching may take advantage of an opportunity for energy savings or performance enhancement with a probability higher than that of coarse grain granularity. Figure 3.1 shows the high level micro-architecture of *AdaCore* architecture. The shaded structures denotes the micro-architecture units that need to be reconfigured dynamically to morph from one core-mode to another.

The buffers that are adaptively resized are re-order buffer, load/store queue and issue queue. Fetch width and issue width are also dynamically resized. Decoding units are subsequently powered on/off when fetch and issue width are resized.

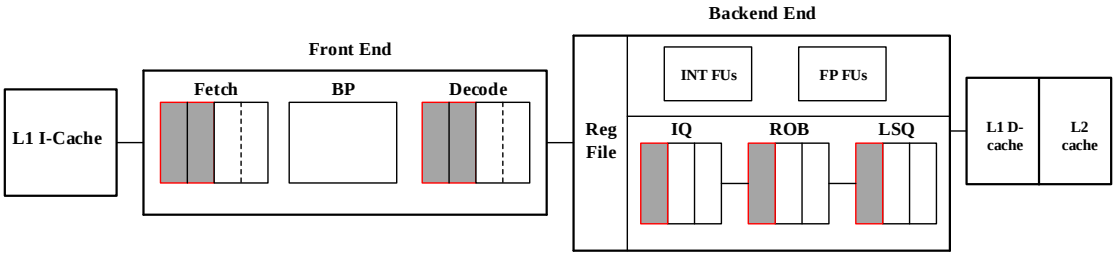

Figure 3.1: High level view of the *AdaCore* Micro-architecture (shaded units represent reconfiguration at run-time).

3.3 Adaptive Storage Buffers

Buffers present in processor store instructions temporarily to overcome timing dependencies that may occur in different stages of the pipeline and also increase the window of instruction to take advantage of instruction level parallelism. The proposed architecture seamlessly moves between four different core configurations thus taking advantage of heterogeneity within. Previous works in [9, 32] have presented adaptive buffer design and have shown significant energy savings when buffers are sized up/down based on the resource requirements of individual applications. However, these works consider individually modifying each buffer wherein they need a independent control mechanism for every buffer.

In this thesis, some of the adaptive buffer schemes presented in [9, 32] are used. The average occupancy in LSQ, ROB and IQ changes rapidly throughout the program execution [9] thus making it difficult to make the right decision on buffer sizing. The scheme implemented in this work is much simpler, since occupancy of LSQ, ROB and IQ are not monitored independently. Determining the transition from one core-mode to another is guided by a function that estimates the power and performance in all core-modes based on performance counter values in the current core-mode where the program is being executed. The *AdaCore* chooses to switch core-modes only when a significantly higher performance/Watt is predicted for another core-mode.

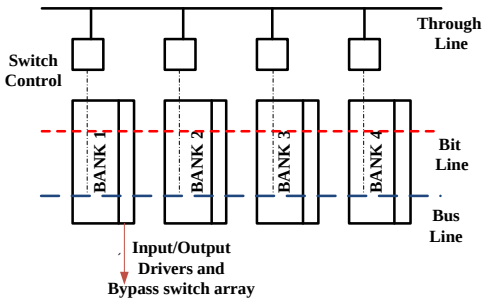

Figure 3.2: Issue Queue Partitioning

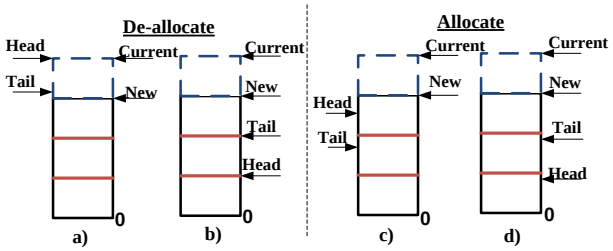

Figure 3.3: ROB Partitioning

ROB, IQ and LSQ are each implemented in a banked architecture similar to the scheme described in [32]. Each banked unit can be independently controlled. The banks have their own set of input/output drivers, precharger and sense amplifiers.

A dynamically re-sizable buffer can be formed by stacking more than one of these partitions together. Figure 3.2 shows how the issue queue is partitioned into multiple banks. The bus line and bit line connections running along multiple banks in the issue queue can be connected through the *through line* via the switch control. This switch control block helps in adding the partition to the current IQ and thereby enabling an increase in the size of the issue queue. To deallocate any partition within the IQ, the switch control could turn off a partition. To simplify the resizing circuitry of the IQ, the IQ needs to be downsized by turning off the partition which has the highest address. Also, before downsizing, we must wait until the existing instructions in the

target partition are issued before switching that particular bank off. Unallocated partitions are powered down which helps in reducing leakage power.

The LSQ and ROB have circular FIFO structures and thus need to be resized carefully. We explain the resizing scenario for ROB shown in Figure 3.3. The *head* and *tail* pointers in ROB give us information about the next instruction to be committed and the location of the next addition to ROB, respectively. The pointer *current* indicates the current upper-bound size for ROB. The *New* indicates the upper bound size for ROB after allocation and deallocation of the partition is done in ROB.

In the ROB deallocation scenario shown in Figure 3.3 (a), both the ROB tail and head pointers are located in the place where the partition is going to be deallocated. So the deallocation will be stalled until the ROB *tail* pointer becomes equal to the *new* pointer and ROB head pointer wraps around itself such that it points to 0. Another case scenario of deallocation is shown in Figure 3.3 (b) when deallocation can happen immediately as both ROB head and tail pointers are not in the bank to be deallocated. While sizing the ROB up, we need to ensure that the head pointer value is below the tail pointer value as shown in Figure 3.3 (d). Allocation is deferred till the ROB head pointer reaches 0 as shown in Figure 3.3 (c).

The partition size for ROB, LSQ and IQ needs to be determined carefully. Determining the optimal partition size is important since too small a partition might result in larger partitioning overhead in terms of layout area and cost. Thus, the partition size for ROB and LSQ is set to be 16 and for IQ the partition is set to be 8 based on [9].

CHAPTER 4

RUNTIME MODE SWITCHING IN *ADACORE*

In the proposed *AdaCore* architecture, one of the most important design decisions to be considered is when to switch from one core-mode to another. In morphable core architectures, the state of the program does not have to be rebuilt on every switch as it would on a heterogeneous multi-core architecture. Therefore, decision to switch between core-modes could be performed at a very fine grain granularity.

Since the core parameters that are modified as part of morphing between core-modes are not restricted to buffers, switching decisions taken based on resource utilization may not be completely accurate. This thesis proposes an online mechanism which estimates power and performance on each of the available core-modes to determine the most efficient core-mode for the phase of application that is currently being executed.

A wrong decision to switch may have a large impact on performance while not contributing enough towards energy savings. Therefore, a switching decision taken based on accurate power and performance estimations is preferred. We present in the next subsection a performance counters (PMCs) based scheme to estimate the power and performance of each of the chosen core-modes with reasonable accuracy.

4.1 Power/Performance Estimation Based on Performance Counters

The decision to select the most appropriate core-mode for the current application phase is made by computing the performance and power for each of the core-mode

and switching the application to run on the core-mode that provides the highest $IPS^2/Watt$ where IPS is Instructions Per Second which is a product of IPC and frequency. Reasons for choosing $IPS^2/Watt$ as our performance/Watt metric is explained in Section 5.1.

To calculate $IPS^2/Watt$, we use PMCs to estimate the power and performance in each of the core-modes. The first step is to find a set of counters that could be used for estimating power and performance and then identify a subset of counters that could be used to estimate both power and performance in runtime. Following are the counters that were considered for performance and power estimation.

1. ***IPC***: The IPC provides the most basic observation point to estimate the amount of power consumed.

2. ***Cache activity***: Lowering the number of cache misses improves the performance of a processor. This applies to both L1 (I/D) and L2 caches. The performance in turn corresponds to power as a miss in the cache leads to additional cycles spent dissipating power. Therefore cache hit/miss/access statistics at level 1 (*L1h, L1m*) and level 2 (*L2h, L2m*) are important.

3. ***Branch activity***: Since inability to predict branches with reasonable accuracy causes performance loss, Branch mis-prediction statistics (*Bmp*) are considered towards estimation.

4. ***Instructions Committed***: The mix of instructions in a given period of time are good indicators of the set of resources utilized. Thus, hardware counters which count the number of Integer *(INT)*, Floating-point *(FP)*, Load *(L)*, Store *(St)* and Branch *(Br)* instructions committed serve to estimate the performance and power consumption.

5. ***Buffer-Full stalls***: Instructions may spend a considerable amount of time in the OOO pipeline due to unavailability of buffer resources (ROB, IQ, LSQ).

Since the time spent stalling in these scenarios impacts performance and in turn power, the stall counters are candidates for the estimation mechanism.

Though in theory, all of the counters described above could be used to obtain performance and power estimations, all of them may not be required to achieve sufficient estimation accuracy at all scenarios. If we could determine the smallest set of counters that would allow us to estimate power/performance accurately on each of the core-modes, some amount of area spent on on-chip counters and computation time. As we consider 4 different core-modes, each core-mode needs to estimate power and performance on all the other 3 core-modes using the hardware counters in the present mode. Power also needs to be estimated on the current core-mode since runtime power measurement is not possible.

For example, if the current application is running on the AC mode, we need to estimate the power of AC mode and the power and performance on the other core-modes that include the narrow core, wider core and larger window core, using the PMCs of the AC mode. Thus, each time a core switching decision is made, the current core-mode needs to make a total of 7 estimations to calculate $IPS^2/Watt$ which would in turn guide the thread to be run on the most efficient core-mode.

To select the PMCs that exhibit the highest correlation with the required estimates (of power and performance) and then obtain the corresponding expressions (using linear regression), we chose a training set of 7 SPEC2000 benchmarks [39] which includes *swim, equake, ammp, applu, twolf. mcf, bzip*, where each of these workloads have application phases that prefer one of the four different core-modes. The workloads were run on all four core configurations for 2 billion instructions after fast forwarding 1 billion instructions. The values of the above described counters are tracked at fine grain instruction granularity, i.e., after every m instructions committed during the execution of the workload. The sensitivity study to choose the value for m is presented later in this chapter.

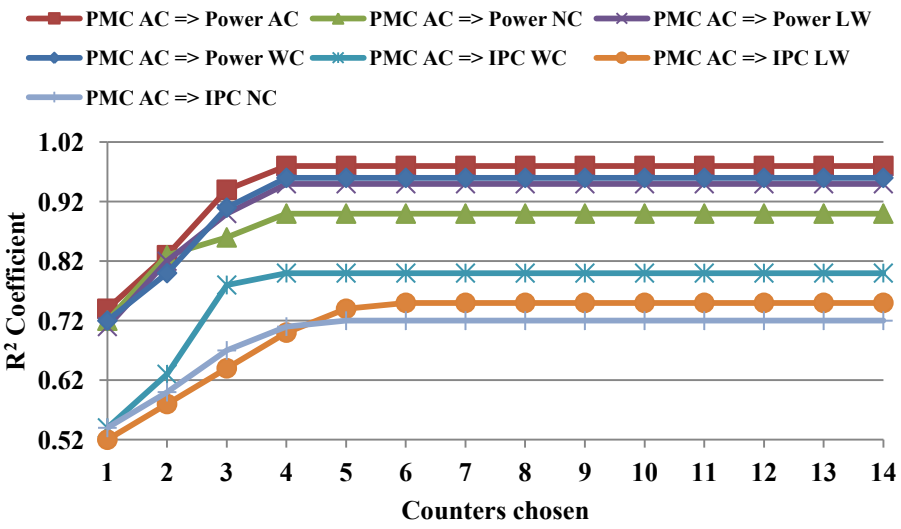

Figure 4.1: R^2 as a function of number of performance counters chosen while estimating power/performance. PMC AC => Power NC, denotes using the performance counters of the AC-mode to estimate the power on the NC-mode.

To find the right counter subset, we iteratively explore all possible combinations of counters to choose the ones that give the highest correlation to the estimated power or performance. At each iteration the counter selection algorithm picks one counter that provides, together with the previously chosen counters, the best fit for estimating power/performance. A linear regression model was used for the counter selection where the best fit was determined by the R^2 correlation coefficient. As shown in Figure 4.1, the R^2 value saturates after a sufficient number of counters is selected and thus, extra counters are no longer added once the value of R^2 starts to saturate.

The number of power/performance calculations made at each decision point is 7. When estimating the power in other core-modes, the R^2 value tends to saturate after adding more than 4 counters. It can also be seen in Figure 4.1 that estimating power in the same core gives a higher R^2 value than in other core-modes indicating higher estimation accuracy. For IPC estimation we need a minimum of four counters to

Table 4.1: Power (P) and performance (IPC) estimation for the other three modes using the performance counters values in the AC mode.

Estimated Param	Expression
AC ⇒ Power AC	$0.31 \cdot L1h + 11.10 \cdot IPC$ $+3.40 \cdot 10^{-2} \cdot St - 2.28 \cdot 10^{-2} \cdot Bmp + 0.29$
AC ⇒ Power NC	$0.12 \cdot Bmp + 0.65 \cdot L1m$ $-0.35 \cdot Br + 1.45 \cdot 10^{-1} \cdot St + 0.64$
AC ⇒ Power LW	$-2.3 \cdot L1m$ $+0.12 \cdot IPC - 3.42 \cdot 10^{-2} \cdot Bmp + 0.35$
AC ⇒ Power WC	$-1.03 \cdot L - 0.76 \cdot Bmp$ $-4.50 \times 10^{-2} \cdot L2h + 0.11$
AC ⇒ IPC WC	$-1.25 \cdot L1m + 1.35 \cdot IPC$ $-1.20 \times 10^{-2} \cdot L2h + 0.23 \cdot Bmp + 0.26$
AC ⇒ IPC LW	$0.14 \cdot IPC - 0.18 \cdot L2h$ $+0.029 \cdot St + 0.11 \cdot Br + 0.56$
AC ⇒ IPC NC	$-0.12 \cdot Bmp + 0.8 \cdot IPC$ $-0.12 \cdot St - 0.11 \cdot L1h + 2.12$

estimate the IPC on the WC-mode and NC-mode but R^2 saturates at five counters for the LW mode.

The selected sets of counters for the estimations are not identical for all the core-modes as shown in Table 4.1. To avoid the need for separate estimation algorithms efforts were made to find a common set of counters which could be used for estimating both performance and power on the AC-mode for the other core modes with reasonable accuracy. Similar analysis was carried out for the estimations on other core modes. We finally arrived at a common set of counters that could estimate performance and power for each of the core modes with reasonable accuracy.

4.1.1 Accuracy of Power/Performance Estimation

The accuracy achieved by our scheme is shown in Figure 4.2. The evaluation of our scheme was performed on a set of 15 workloads which contain a mix of SPEC2000 and SPEC2006 workloads. Due to larger set of estimation scenarios, we show only the average error in estimating the power and IPC on each of other core-modes using the PMCs of the current core-mode. It could be observed from Figure 4.2 that the

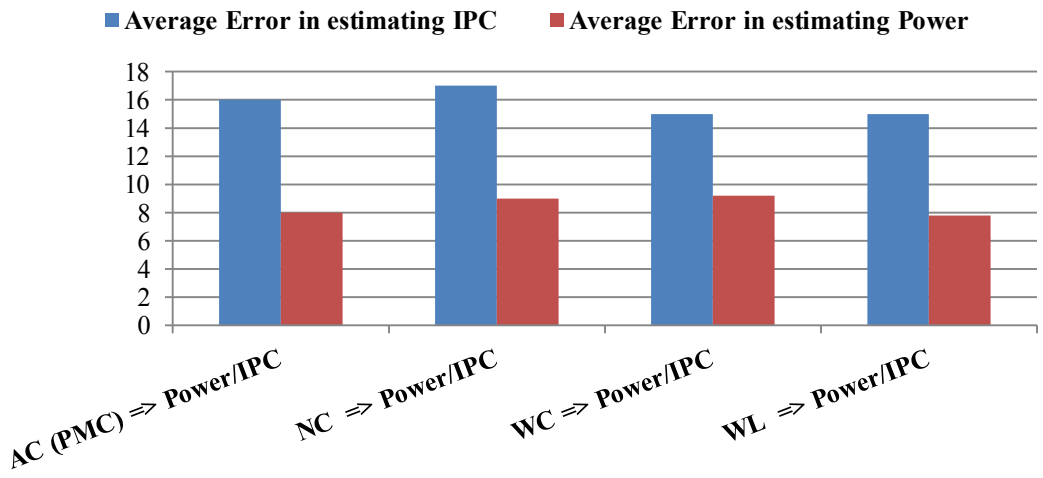

Figure 4.2: Average error in estimating power and IPC for each core-mode.

average error in estimating power is around 8% which is significantly lower than the average error in estimating IPC which is around 16%.

Though the overall estimation error is low, the scheme implemented could make wrong decisions if the estimation error at runtime is high. An analysis of the distribution of error is shown in Figure 4.3. As IPC had higher average error than power, for sake of brevity, we only show the error in estimating IPC in all other core-modes using the PMCs of the current AC-mode in Figure 4.3. We can see that deviation of error from the mean is low for majority of sample points with 80% of sample points lie between tolerance level of 10% from the mean. Thus, we can conclude that we can predict both power and performance with adequate accuracy for each of the core-modes.

Using the power and performance estimated online, we compute $IPS^2/Watt$ for each of the core-mode. The decision to select the core-mode is based on the core-mode which provides us the highest $IPS^2/Watt$. But while taking the decision to switch, it is important that we remain in a particular core-mode for sufficient amount of time before switching back to another core type. Since frequent switching would result in

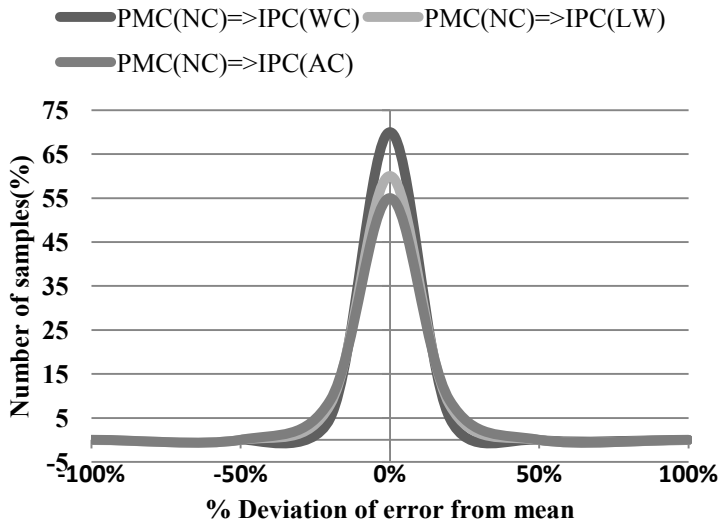

Figure 4.3: Distribution of error in estimating IPC in different core-modes using the PMCs of the current mode.

increased overhead and negate the benefits of the proposed scheme. Therefore, it is important to determine the confidence of the decision as described in the following section.

4.2 Decision Granularity

The decision to morph into any of the three available four core-modes or to remain in the current core-mode is taken every time a certain number of instructions are committed. This threshold on the number of instructions committed per decision is termed as 'window'. The window length is denoted by l. Since an overhead is involved in mode switches which affects processor performance, a switch in the core-mode takes place only if the same decision to switch is repeated for k successive windows termed as 'history depth'. Therefore, a decision to switch is taken only if the decision is consistent over a period of time during which m instructions are committed where, $m = k \times l$. For example, if for the past m committed instructions, the decision to move from the current mode (AC-mode) to NC-mode was repeated for k successive

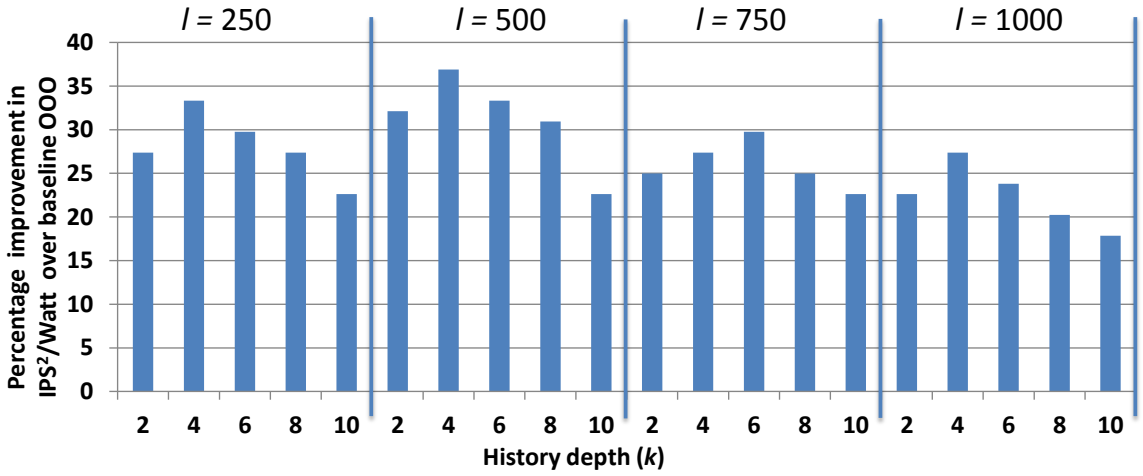

Figure 4.4: Improvement in $IPS^2/Watt$ of *AdaCore* over baseline AC for multiple window length, l and m combinations

windows, a prediction that the workload has entered a phase where NC-mode provides higher $IPS^2/Watt$ is made. The window size and value of k need to be determined experimentally. A sensitivity study has been conducted to quantify the impact of window length and k on the achieved benefits.

The l and k combination that yields the highest $IPS^2/Watt$ for the entire program execution would be the best choice. A window length from 250 to 1000 instructions in steps of 250 has been experimented with. For a particular window size, the value of k is varied from 2 to 10 in increments of 2. To determine the best l and k, we ran a set of 10 workloads. We computed the percentage increase in $IPS^2/Watt$ of our proposed core over running the application on the baseline AC core-mode.

As observed from in Figure 4.4, window length, l of 500 and k of 4 provide the highest improvement in $IPS^2/Watt$.

4.3 Adaptive Voltage and Frequency Scaling in *AdaCore*

In *AdaCore*, each of the core-modes runs at different voltage and frequency levels. Thus, while switching from one core-mode to another, we need to make sure that

overhead due to voltage and frequency scaling is not large enough to negate the benefits obtained from our proposed morphing scheme.

Whenever there is change in processor frequency, the PLL needs to re-lock to the new frequency which would result in halting of the processor. This overhead in Intel's processor is claimed to be 5 μs [30]. There is also additional overhead involved 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. This overhead is estimated to be around 25μs for the range of voltage/frequency considered based on the work in [30]. Thus, to reduce the high overhead, frequency scaling has been applied in the past only at coarse grain instruction granularity in the order of millions of processor cycles.

In our proposed scheme, we morph from a core-mode with a particular voltage/frequency(v/f) setting to a core-mode with a different (v/f) values at a fine-grain instruction granularity. Recently, Kim *et al.* studied the use of an on-chip regulator that would allow scaling voltage on the order of tens of nanoseconds or hundreds of processor cycles [19], [20], [25]. Their observation was that with the help of on chip regulator, voltage and frequency scaling could be performed at fine-grain at the order of hundreds of processor cycles. Hardware based fine-grain voltage and frequency scaling mechanism with an on chip regulator was implemented by Eyerman *et al* where switching v/f modes was made possible upon individual off chip memory access [10].

Thus, in this work, fine grain voltage and frequency scaling is used in transition from one core-mode to another with an estimated overhead of 200 cycles.

4.4 Decision Controller and Overheads

To enable morphing between different core-modes, we need a controller that interacts with the core and helps the core to make smooth transition from one core mode

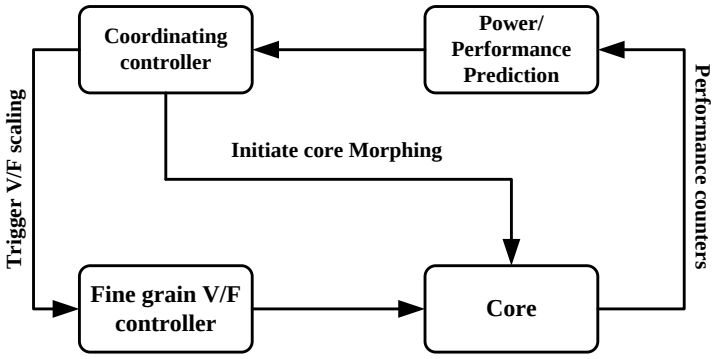

Figure 4.5: *AdaCore* Decision Controller framework

to another. As shown in Figure 4.5, the controller consist of power/performance estimator which receives the performance counters (PMC) values sampled at fine-grain instruction granularity from the core. Once it receives the PMC values, it computes power and performance on each core-mode, using the trained expressions obtained offline as explained previously. The computed values are then fed into the coordinating controller which then computes $IPS^2/Watt$ on each of the core-modes and determines the best core to morph into.

If the computed $IPS^2/Watt$ value on the other core-mode is greater than the mode it is currently executing and the difference is greater than a fixed determined threshold, the controller initiates the morphing in *AdaCore*. When core reconfiguration is initiated, core parameters of the currently executing core consisting of ROB, LSQ ,IQ, fetch and issue width are resized. The coordinating controller also needs to interacts with fine grain v/f controller to change the core voltage and frequency only when a decision to switch is initiated.

If the controller determines that the best core-mode for next interval of execution is the current core-mode, the fine grain v/f controller need not be initiated as there is no change in voltage or frequency. Previously proposed morphable architectures use a similar kind of controller based feedback technique to guide the core to the

right mode where overhead on each performance and power estimation to compute $IPS^2/Watt$ was estimated to be 30 cycles [27].

The fine-grain voltage and frequency scaling overhead is assumed to be 200 clock cycles as mentioned earlier. 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 [40]. 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. 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 impact.

CHAPTER 5

EVALUATING *ADACORE*

In this chapter, an evaluation of the proposed morphable architecture, *AdaCore* is presented. We use gem5 as our cycle accurate simulator with integrated McPAT modelling framework to compute power of the cores and L1 caches [4, 38]. We evaluate our proposed scheme with SPEC2006 [5] and SPEC2000 [39] benchmark suites. The benchmarks were compiled using gcc for Alpha ISA with -O2 optimization. All the evaluations were carried out by running benchmarks for 2 billion after skipping the first 2 billion instructions. Below, we discuss the evaluation metric and compare to various other morphable architectures.

5.1 Selection of Switching Metric

Switching from one core-mode to another is based on the energy-delay product metric (EDP). The EDP metric tries to capture the importance of both energy and performance in a single expression. Since each of the modes uses a different frequency, we try to minimize the energy-delay product at each window interval by morphing into a core-mode that would provide the highest $IPS^2/Watt$ for that interval. If we increase the weightage of the performance in the metric as in $IPS^3/Watt$ (related to energy $\times$ $delay^2$) or $IPS^4/Watt$, these metrics would try to find core-modes that provides best performance. Since our objective is to maximize the energy efficiency without considerable performance loss or may be even with performance gain, we select $IPS^2/Watt$ as our switching threshold metric. Thus, the decision to morph from the current core-mode to another is taken if the $IPS^2/Watt$ computed for each

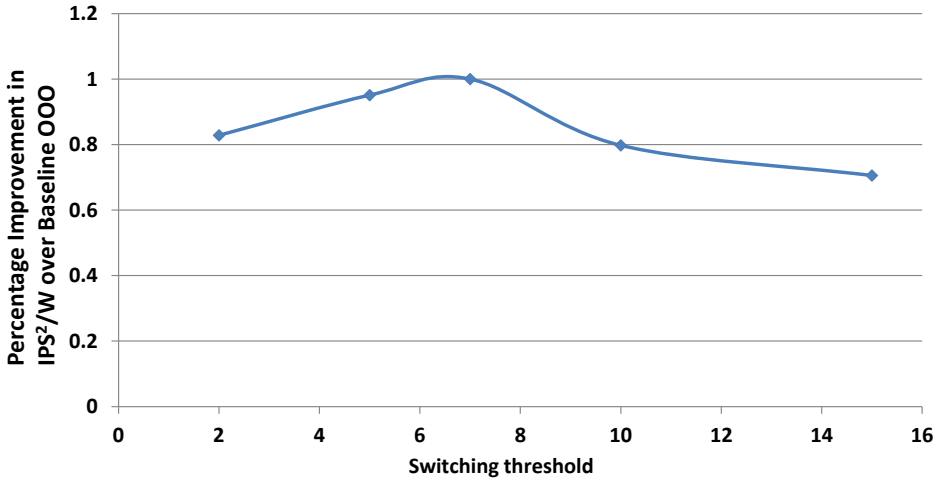

Figure 5.1: $IPS^2/Watt$ improvement for *AdaCore* architecture with online mode management scheme over baseline single OOO core as a function of switching threshold.

of the other core-mode is greater than that of the current mode by atleast a certain threshold that needs to be determined. Figure 5.1 shows that the improvement in $IPS^2/Watt$ over the baseline is largest when the switching threshold is 7%. If more than one core-mode is predicted to gain more than 7% of $IPS^2/Watt$, the mode with highest $IPS^2/Watt$ is chosen as our core-mode which we would morph into.

5.2 Performance/Watt of *AdaCore*

The percentage of time spent by various workloads in each of the core-modes is shown in Figure 5.2 which justifies the need for a four-core-mode morphable architecture that can address varying resource demands in diverse application phases. At fine-grain granularity when low performance phases exist, these phases could be mapped to a core-mode which is suited to target these low ILP phases.

The percentage improvement in $IPS^2/Watt$ of the proposed *AdaCore* when compared to the baseline where application is run completely in OOO core is shown in Figure 5.3. On average, we achieve an $IPS^2/Watt$ improvement of 39% and per-

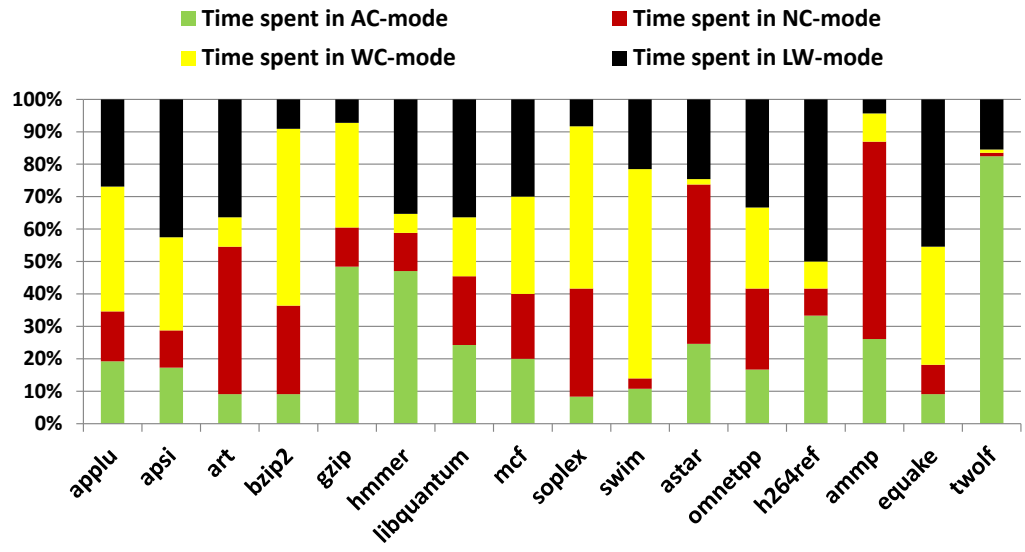

Figure 5.2: Time spent in each of the *AdaCore* modes by SPEC workloads.

formance improvement of 10% when compared to the baseline. Benchmarks which are memory intensive or have high branch mis-prediction rates such as *mcf, soplex, astar, ammp* achieve higher improvement in $IPS^2/Watt$ due to presence of more low performance unstable phases that exist at fine grain granularity which could take advantage of the proposed morphable *AdaCore* architecture. Compute intensive benchmarks such as *hmmer, bzip2, h264ref* also take advantage of *AdaCore* and provide substantial benefits using core-modes that can relieve bottlenecks present in the application phases. Figure 5.4 shows the average performance gain obtained in *AdaCore* fine-grain sampling, *AdaCore* coarse-grain and *AdaCore* with Oracle prediction schemes for different values of the thread switching overhead. Even if the overhead of mode switching is to increase from 500 cycles to 5000 cycles, the average performance gain is reduced by only 4% for the proposed reconfiguration scheme.

The key difference between *AdaCore* and existing morphable architectures with two core-modes consisting of OOO/InO [27] is that, the latter is more useful to applications which have more memory intensive phases or phases with high branch mis-prediction rates. For example, benchmarks such as *mcf, astar, soplex* which

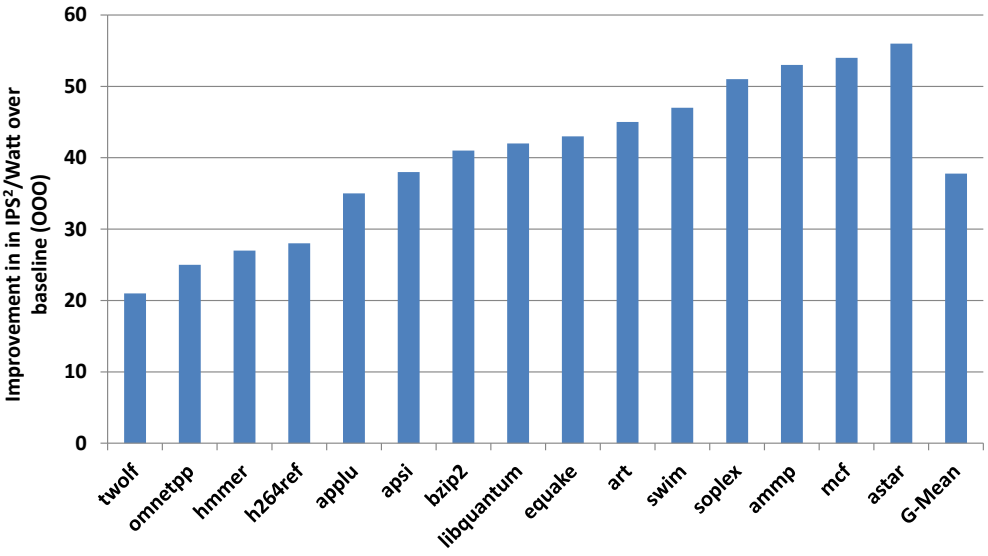

Figure 5.3: $IPS^2/Watt$ improvement achieved by the *AdaCore* architecture with online mode management scheme over the baseline single OOO core.

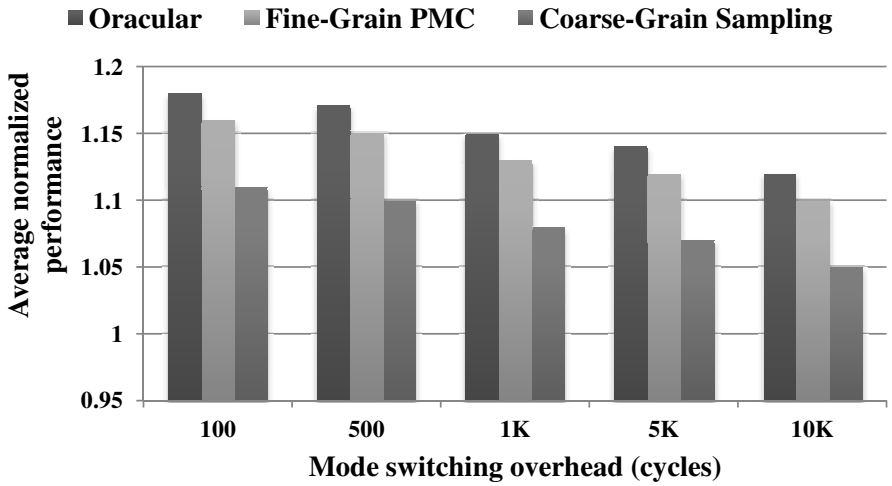

Figure 5.4: Impact of switching overhead.

contain these phases can be mapped to energy efficient in-order core. The energy efficiency obtained in mapping to in-order core comes with performance loss. Compute intensive benchmarks benefit very little from these morphable architectures as they have very few phases with low performance that could be mapped to the in-order core, thus resulting in insignificant gain in energy efficiency.

For morphable architectures like *AdaCore*, a diverse set of application phases such as memory intensive phases, phases with high branch miss-prediction or compute intensive phases, can benefit from having core-modes designed to resolve application performance bottlenecks. The proposed morphable architecture is also able to extract energy saving opportunities available at fine grain granularity. Figure 5.5 indicates that *AdaCore* achieves a 35% improvement in energy savings over a static OOO core.

5.3 Comparison with other schemes

A comparison between the proposed *AdaCore* and various other schemes or architectures is presented below.

5.3.1 Fine-grain *AdaCore* vs Coarse-grain *AdaCore*

AdaCore being a morphable architecture can alleviate processor bottlenecks faster than an HMP architecture which requires transfer of state to another core. Therefore, *AdaCore* was originally envisioned to switch at a fine-grain granularity. However, an important question to consider is, *would switching modes at a coarse grain granularity provide the same amount of benefits as switching at fine-grain?*.

To this end, we implement two versions of *AdaCore*, i.e., Fine-grain *AdaCore* vs Coarse-grain *AdaCore*. For coarse grain switching, a sampling based technique has been used. To model sampling based algorithm we use two parameters, the switching interval and sampling interval as introduced in [28]. To make a decision regarding mode switching, after every switching interval the application is sampled on all of

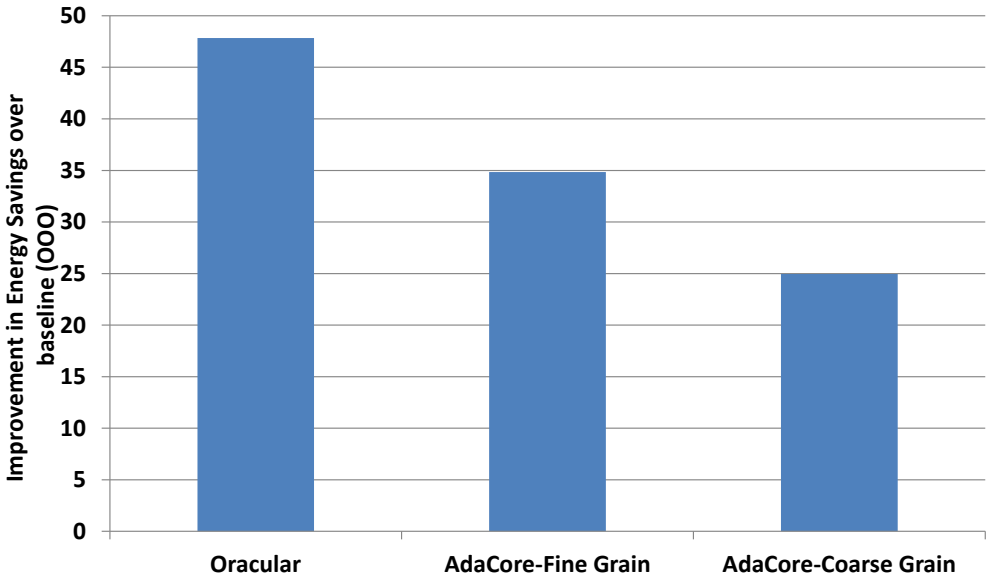

Figure 5.5: Energy savings over baseline for variants of *AdaCore*

the available modes. The best core-mode which is determined during the sampling interval is the mode where the application is run for the next switching interval. The switching interval is taken to be 1M with sampling interval of 10K instructions [28].

Energy savings obtained by the schemes over a static baseline OOO core are shown in Figure 5.5. *AdaCore-coarse grain* provides 25% energy savings which is 10% less than our *AdaCore* with four core-modes and a fine-grain scheme guided by performance counters to make switching decision. Lower energy savings obtained from sampling based schemes arise due to incorrect decision in finding the right core mode and inability to take advantages of opportunities that occur at finer granularity.

5.3.2 *AdaCore* vs HMP with Non-monotonic core-types

In this section, *AdaCore* architecture is compared with the HMP architecture proposed in [28] consisting of non-monotonic core types as shown in Table 3.1. Navada *et al.* [28] identify bottlenecks in the current core using performance counters and transfer the current application to the core that can relieve this bottleneck. Since the cores are independent, upon each swap of application from one core to another they

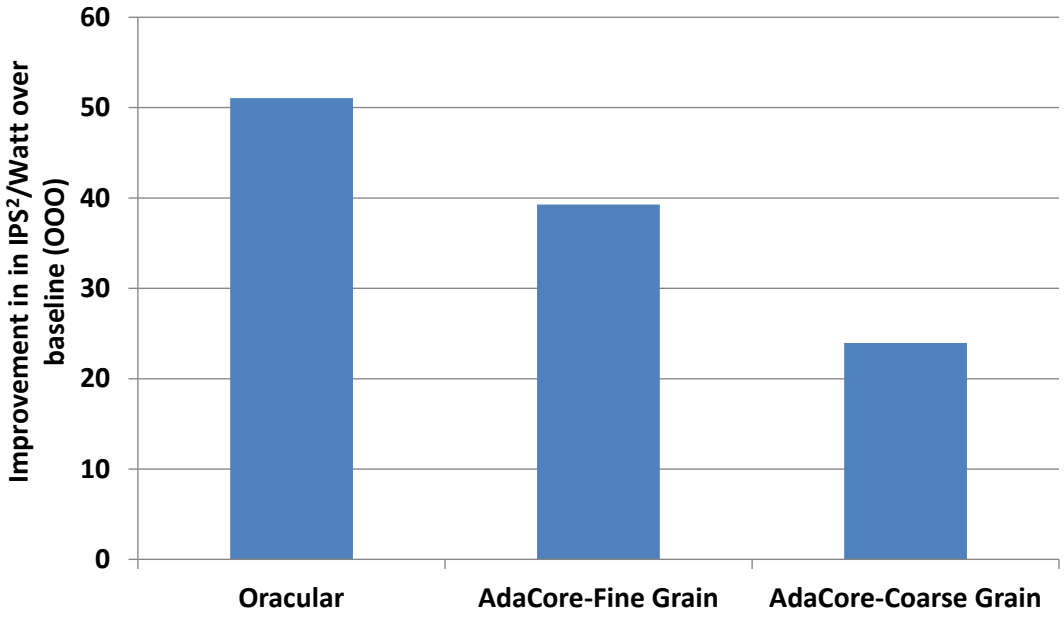

Figure 5.6: Improvement in IPS^2/Watt over baseline OOO core for multiple *AdaCore* schemes.

need to account for register transfer overhead and L1 cache misses. As a result their architecture does not allow to perform thread switching at a fine granularity. Their decision to switch is taken after every 10K instructions.

Our proposed morphing scheme can achieve 15% more improvement in IPS^2/Watt over an HMP with identical core-types. Higher IPS^2/Watt is achieved due to more reconfigurations that take place at lower instruction granularity as shown in Figure 5.7 for *AdaCore* with four core-modes. At lower granularity there is potential for more lower performance phase intervals thus providing more opportunities to use energy efficient mode resulting in increased switching activity as shown in Figure 5.7. For our instruction length of 2K we obtained about 500 more switches/million instructions when compared to the 10K interval resulting in a higher performance/Watt. For the switching threshold of 7%, for every 10M instructions committed, it was found that *AdaCore* switches modes with a probability of 17 percent.

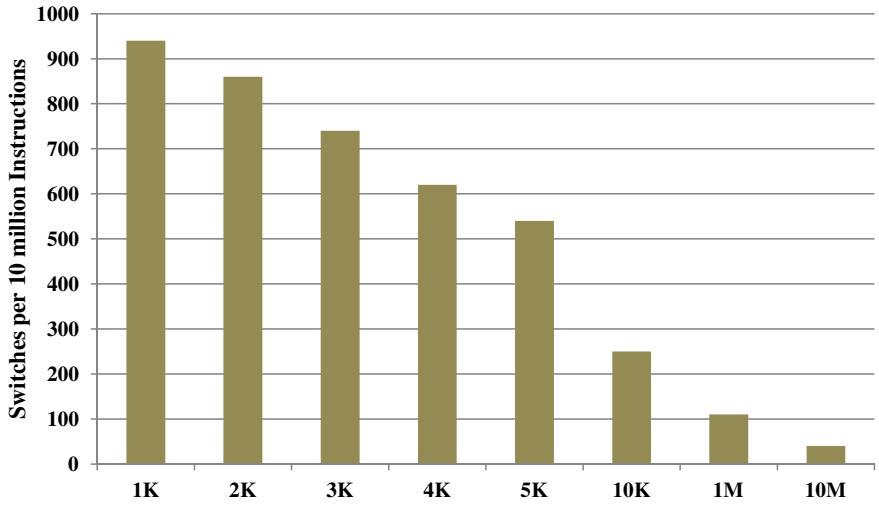

Figure 5.7: Number of switches per ten million instructions for a range of instruction granularities.

5.3.3 *AdaCore Oracle* vs *AdaCore PMC*

In the *AdaCore Oracle* scheme, an oracle is used to steer the application to the right core-mode. Switching between core-modes is performed at instruction granularity of 2000 as determined earlier. Thus for every 2K instruction granularity, the oracle chooses the core that best suits the application. Energy savings obtained from the oracular scheme is shown in Figure 5.5. The oracular scheme provides 12% more energy savings than our *PMC-AdaCore*. It was observed that 6% of all the switches made by *AdaCore PMC* differed from switches suggested by the *Oracle*. Further analysis revealed that for 2.5 percent out of the 6 percent, *AdaCore PMC* switched to a mode which was not suggested by the oracle while the remaining 3.5 percent of switches were not required at all. As this scheme is not practical, it provides an upper-bound for maximum energy savings that could be achieved from a morphable architecture with four core-modes.

CHAPTER 6

COMPARATIVE STUDY WITH DYNAMIC VOLTAGE AND FREQUENCY SCALING MECHANISM

Several DVFS schemes which achieve multiple and wide range of objectives were presented in chapter II. In this chapter, we study whether a superscalar OOO processor can achieve energy efficiency similar to that of the morphable architectures while losing as little performance as possible by efficiently utilizing DVFS at a fine grain granularity.

The trigger to scale the frequency and/or voltage in the processor could vary depending on the objective of the scaling. The applications running on a standard processor may not exhibit the same behavior during the entire execution. Their behavior, resource demands and performance may vary over time. A workload section exhibiting similar characteristics over a short period of time is termed as a phase. Therefore, an application may include multiple phases over the entire benchmark execution. Individual phases may perform best at distinct voltage and frequency points.

Researchers have found that the applications in SPEC 2000 and SPEC 2006 benchmarks display wide range of behavior at a small granularity of very few thousands of instructions [31]. Therefore, *AdaCore* and the DVFS scheme that we compare, attempt to respond to changing application phases by switching a core mode or (v/f) mode every few thousand instructions. The core considered to use DVFS is identical to the Average core (AC) from Table 3.1. The frequency is varied from 1.2GHz to 2GHz in steps of 200 MHz. It is to be noted that the voltage and frequency points

Table 6.1: Voltage and Frequency modes considered.

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

picked in Table 6.1 are taken considering 32nm CMOS process technology. The voltages for the corresponding frequencies were selected in accordance with Intel's 32nm datasheet [14].

Traditionally, scaling the operating voltage is performed by using off-chip regulators. The overhead of using off-chip regulators often run into tens of thousands of cycles [6]. Such high overhead limits the granularity at which DVFS can be performed. Clearly, scaling voltage and frequency dynamically at finer granularities can provide greater energy efficiency for processors with very small overhead for scaling. Therefore, Kim *et al* dedicated research efforts into designing a low overhead on-chip voltage regulator [20]. The overhead to switch the v/f mode at runtime was reduced to 200 cycles [10]. The average additional latency in case of *AdaCore* was assumed to be 300 cycles in addition to the 200 cycles for DVFS scheme alone. As mentioned previously, the outcome of DVFS or morphable architectures is highly dependent on the mechanism which determines the mode switching in either scheme. The runtime switching mechanism considered in both schemes are described in the next section.

6.1 Runtime V/F Mode Selection

6.1.1 Simulation and Oracular Study

To simulate the DVFS described above, the baseline AC core with DVFS capability is simulated using gem5 [4]. The power over a period of workload execution is obtained using the McPAT power simulator [38]. A set of 15 SPEC2000 and 2006 workloads

were run through the cycle accurate simulator for 2 billion instructions after fast forwarding a billion instructions. The window length and m from Section 4.2 were used. Then, performance and power over the execution of each of the workloads were collected. An oracle steering algorithm is utilised to arrive at the best v/f mode to run on at each instant of time. The oracle is aware of power and performance of the OOO core in each of the v/f modes and it chooses the mode that achieves the best value of the decision metric. Following is a study which was used to choose the decision metric.

To compare the DVFS scheme with *AdaCore*, the proposed DVFS scheme is to be designed to achieve the highest power efficiency while losing only little in performance. Therefore, the decision metric used to drive the DVFS needs careful consideration. The DVFS enabled core is capable of scaling the frequency and voltage to any of the modes mentioned in Table 6.1. The decision to scale the frequency is performed based on the decision metric $IPS^n/Watt$ where n is a positive number. To determine the value of n, several DVFS experiments with an oracle steering algorithm were conducted. In each of the experiments, the oracle determined the best v/f mode based on the decision metric $IPS^n/Watt$ where n is a value between 1 and 3. The IPS/Watt, IPS and power improvement normalised to the baseline are presented in Figure 6.1 for selected values of n.

It is observed from Figure 6.1 that for n=1, we obtain a 38% improvement in power efficiency measured by $IPS/Watt$ but incur a performance loss of about 16% over the baseline mode. We observe that the $IPS/Watt$ gain for n=2 is 26% with 6% loss in performance. Since we see minimal benefits for n=3, we search for a value of n between 2 and 3. The aim is to obtain the highest $IPS/Watt$ while losing minimum performance. From the points shown, we find n=2.2 to be the pareto optimal point and therefore choose n=2.2 which provides 20% improvement in power efficiency with only 3.5% loss in performance.

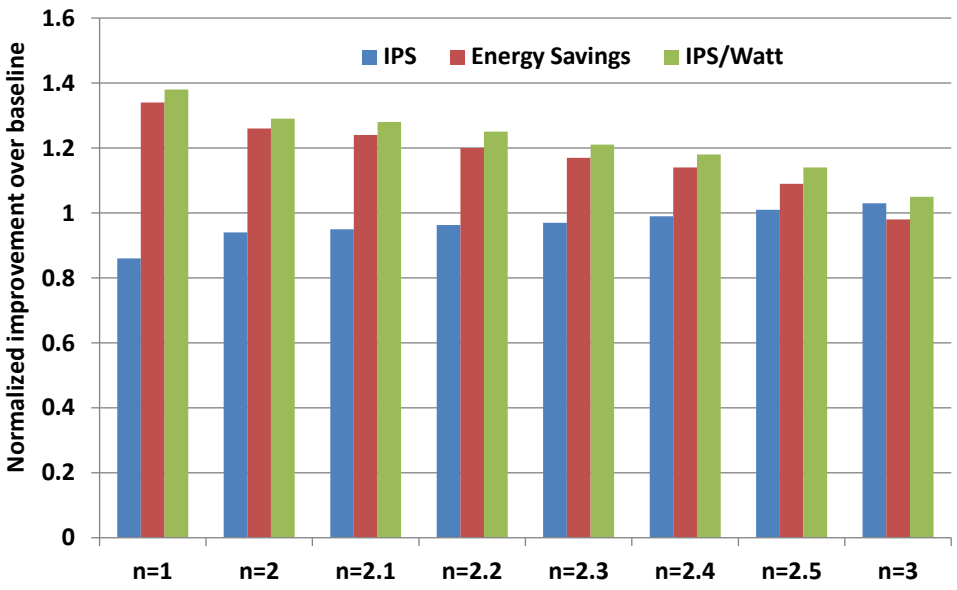

Figure 6.1: Performance, energy savings and performance/Watt improvement over baseline OOO for decision metric $IPS^n/Watt$

6.1.2 Performance and Power Prediction

To compute $IPS^{2.2}/Watt$, for each of the v/f modes, the performance and power on the corresponding modes has to be estimated. The performance and power estimations are similar to that of the *AdaCore* described in Section 4.1. Performance counters collected on the current v/f mode are used as a means to estimate the performance and power in all the other v/f modes using the statistical method explained in Section 4.1.

We use a training set of seven SPEC2006 and SPEC2000 benchmarks to be run on each of the v/f modes. The training set is fast forwarded for 2 billion instructions and run for 2 billion instructions during which the above listed performance counters are collected at a certain instruction granularity. Along with the performance counters, power and performance for each of the runs were also collected through our simulation framework. A linear regression model is used to extract the equations needed for the estimation. The counters selected for prediction are similar to those presented in

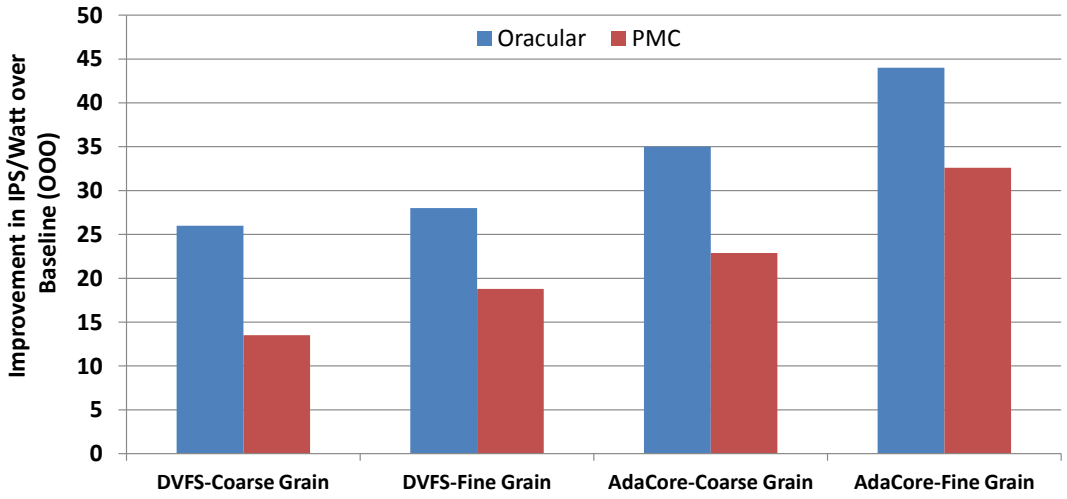

Figure 6.2: Comparison of Fine Grain DVFS scheme with other schemes

Section 4.1. Performance and power estimation equations similar to Table 4.1 were extracted for transitions from each of the modes in Table 6.1.

6.1.3 Fine-Grain DVFS vs Coarse-Grain DVFS

As explained above, given the low overhead of on-chip regulators, processors can afford to switch modes quite frequently. Therefore, the DVFS runtime mechanism considered in this study can afford to switch v/f modes at a very fine instruction granularity. However, an important question to consider is, *would switching v/f modes at coarse grain granularity achieve the same power efficiency as fine-grain granularity.*

To this end, two versions of the proposed DVFS, i,e., *fine-grain and coarse grain DVFS* have been implemented and an evaluation of power efficiency for switching v/f modes at both coarse and fine granularities was performed. In coarse grain DVFS, the decision to switch v/f modes is taken every 1M instructions, while in fine grain granularity, switching decisions are made every 2000 instructions similar to the *AdaCore* evaluation. Figure 6.2 shows that while using both oracular steering algorithm

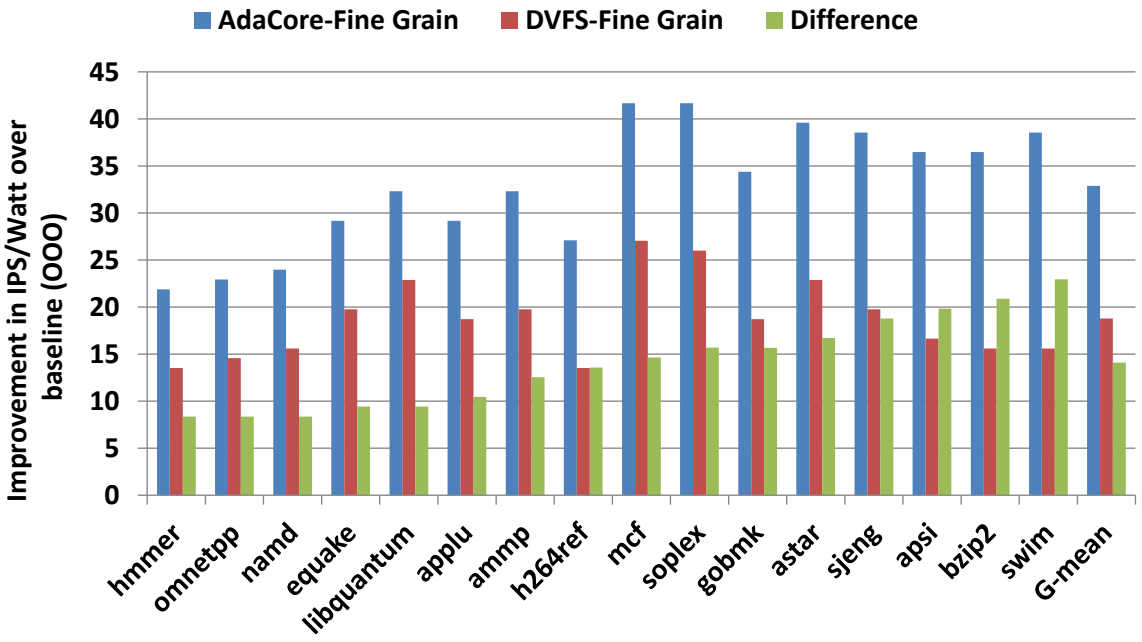

Figure 6.3: Performance/Watt comparison between FineGrain DVFS and *AdaCore*

or a PMC based runtime mechanism, the fine-grain DVFS outperforms coarse grain DVFS in terms of IPS/Watt by 5%.

6.1.4 DVFS vs *AdaCore*

The AC core with features to scale voltage and frequency at runtime at a fine grain granularity was simulated. Multiple v/f modes were selected and a DVFS steering mechanism was developed to improve processor power efficiency. An important question is whether a processor with fine-grain DVFS capability can achieve power efficiency comparable to that of a reconfigurable architecture such as *AdaCore*.

To this end, Figure 6.2 shows the power efficiency (IPS/Watt) achieved by *AdaCore* and Fine-Grain DVFS scheme. It is observed that *AdaCore* outperforms Fine-Grain DVFS by 14%. Higher performance/Watt is achieved by *AdaCore* due to resource reconfiguration at lower instruction granularity.

A benchmark comparison of the fine-grain DVFS and *AdaCore* is shown in Figure 6.3. It illustrates that memory bound benchmarks such as *mcf, libquantum and soplex* exhibit a large IPS/Watt improvement over the baseline as predicted. Higher frequency may not provide high performance gains for memory bound benchmarks. Therefore, they benefit in terms of IPS/Watt when they are run on the low frequency modes. However, we notice a larger IPS/Watt improvement for *AdaCore* as reconfiguring the resources contributes to larger IPS/Watt than fine-grain DVFS alone. In case of branch intensive workloads like *astar, gobmk, swim and sjeng*, we also see a greater than 15% improvement in IPS/Watt for fine grained DVFS. The *AdaCore* achieves a larger improvement for as its smaller resource modes improve power efficiency as was in case of memory bound applications. The compute bound applications like *hmmer, equake and h264ref* however, gain less than 15% in power efficiency through DVFS as compared to greater than 20% observed for *AdaCore*.

CHAPTER 7

CONCLUSION

Since applications experience diverse phase behavior during their execution, they can take advantage of a morphable core architecture.

A morphable architecture has been proposed in this thesis where a single OOO core morphs or reconfigures into three other OOO modes with capabilty to address various processor bottlenecks. The proposed morphable core can adapt to a specific core-mode depending on phase resource demands. The decision to reconfigure is taken at runtime thereby alleviating the loss in performance while also saving energy where there is lack of demand for certain resources, by using aggressive power gating and DVFS techniques. This research work proposes aggressive mode switching at a fine-grain granularity to address the resource demands or lack thereof, to save energy while not losing performance. Therefore, we also provide an efficient run time performance counters based switching decision mechanism that can map the application to the right core-mode at fine grain granularity, thus providing high performance/Watt throughout the application execution.

Multiple versions of the proposed morphable architectures have been considered for comparing with the proposed *AdaCore* with a runtime control mechanism based on performance counters. A comparison evaluation of power efficiency with *AdaCore* and fine-grained DVFS schemes was performed. Our results indicate that we can achieve average performance/Watt benefit of 32% over executing on a static OOO core with performance gain close to 10% and almost 14% performance/Watt over fine-grained DVFS.

BIBLIOGRAPHY

[1] Annamalai, A., Rodrigues, R., Koren, I., and Kundu, S. An opportunistic prediction-based thread scheduling to maximize throughput/watt in amps. In *Proceedings of the 22nd international conference on Parallel architectures and compilation techniques* (2013), IEEE Press, pp. 63–72.

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

[3] Becchi, M., and Crowley, P. Dynamic thread assignment on heterogeneous multiprocessor architectures. In *Proceedings of the 3rd conference on Computing frontiers* (2006), ACM, pp. 29–40.

[4] Binkert, et al. The gem5 simulator. In *ACM SIGARCH Computer Architecture News* (Aug 2011), vol. 39, pp. 1–7.

[5] Bird, S, et al. Performance characterization of spec cpu benchmarks on intel's core microarchitecture based processor. In *SPEC Benchmark Workshop* (January 2007).

[6] Burd, T.D, Pering, T., Stratakos, A. J., Brodersen, R. W., et al. A dynamic voltage scaled microprocessor system. *Solid-State Circuits, IEEE Journal of 35*, 11 (2000), 1571–1580.

[7] Charles, J., Jassi, P., Ananth, N. S., Sadat, A., and Fedorova, A. Evaluation of the intel® core i7 turbo boost feature. In *Workload Characterization, 2009. IISWC 2009. IEEE International Symposium on* (2009), IEEE, pp. 188–197.

[8] Craeynest, Van, et al. 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.

[9] 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.

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

[11] Greenhalgh, P. Big.little processing with arm cortex-a15 and cortex-a7.

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

[13] Horowitz, M., Alon, E., Patil, D., Naffziger, S., Kumar, R., and Bernstein, K. Scaling, power, and the future of cmos. In *Electron Devices Meeting, 2005. IEDM Technical Digest. IEEE International* (2005), IEEE, pp. 7–14.

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

[15] Kejariwal, A., Veidenbaum, A.V., Nicolau, A., Tian, X., Girkar, M., Saito, H., and Banerjee, U. Comparative architectural characterization of spec cpu2000 and cpu2006 benchmarks on the intel core 2 duo processor. In *Embedded Computer Systems: Architectures, Modeling, and Simulation, 2008. SAMOS 2008. International Conference on* (july 2008), pp. 132–141.

[16] Khan, O., and Kundu, S. 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.

[17] Khubaib, Suleman, M. A., Hashemi, M., Wilkerson, C., and Patt, Y. 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.

[18] Kim, C., Sethumadhavan, S., Govindan, M. S., Ranganathan, N., Gulati, D., Burger, D., and Keckler, S. W. Composable lightweight processors. In *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture* (2007), pp. 381–394.

[19] Kim, W., Brooks, D., and Wei, G. A fully-integrated 3-level dc-dc converter for nanosecond-scale dvfs. *Solid-State Circuits, IEEE Journal of 47*, 1 (2012), 206–219.

[20] Kim, W., et al. 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.

[21] Koufaty, D., Reddy, D., and Hahn, S. Bias scheduling in heterogeneous multi-core architectures. In *Proceedings of the 5th European conference on Computer systems* (2010), ACM, pp. 125–138.

[22] Kumar, R., Farkas, K. I., Jouppi, N. P., Ranganathan, P., and Tullsen, D. 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.

[23] Kumar, R., Tullsen, D. M., and Jouppi, N. P. Core architecture optimization for heterogeneous chip multiprocessors. In *Proceedings of the 15th International Conference on Parallel Architectures and Compilation Techniques* (New York, NY, USA, 2006), PACT '06, ACM, pp. 23–32.

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

[25] Le, H., Crossley, J., Sanders, S., and Alon, E. A sub-ns response fully integrated battery-connected switched-capacitor voltage regulator delivering 0.19 w/mm 2 at 73% efficiency. In *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2013 IEEE International* (2013), IEEE, pp. 372–373.

[26] Li, Y., Skadron, K., Brooks, D., and Hu, Z. Performance, energy, and thermal considerations for smt and cmp architectures. In *High-Performance Computer Architecture, 2005. HPCA-11. 11th International Symposium on* (2005), IEEE, pp. 71–82.

[27] Lukefahr, A., Padmanabha, S., Das, R., Sleiman, F. M., Dreslinski, R., Wenisch, T. F., and Mahlke, S. 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.

[28] Navada, S., Choudhary, N. K, Wadhavkar, S. V, and Rotenberg, E. 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.

[29] Olukotun, K., Nayfeh, B. A., Hammond, L., Wilson, K., and Chang, K. The case for a single-chip multiprocessor. *ACM Sigplan Notices 31*, 9 (1996), pp. 2–11.

[30] P., Jaehyun, S., Donghwa, C., 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.

[31] Padmanabha, S., Lukefahr, A., Das, R., and Mahlke, S. 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.

[32] Ponomarev, D., Kucuk, G., and Ghose, K. 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.

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

[34] 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.

[35] Rodrigues, R., et al. 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), pp. 5:1–5:23.

[36] Semeraro, G., Magklis, G., Balasubramonian, R., Albonesi, D. H., Dwarkadas, S., and Scott, M. L. Energy-efficient processor design using multiple clock domains with dynamic voltage and frequency scaling. In *High-Performance Computer Architecture, 2002. Proceedings. Eighth International Symposium on* (2002), IEEE, pp. 29–40.

[37] Shelepov, D., et al. Hass: a scheduler for heterogeneous multicore systems. *SIGOPS Oper. Syst. Rev. 43* (April 2009), pp. 66–75.

[38] Sheng, et al. Mcpat: An integrated power, area, and timing modeling framework for multicore and manycore architectures. In *Proc. of the 42nd Annual International Symposium on Microarchitectur* (2009), pp. 469–480.

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

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

[41] Srinivasan, S., Kurella, N., Koren, I., and Kundu, S. Exploring heterogeneity within a core for improved power efficiency.

[42] Suleman, M. A., Mutlu, O., Qureshi, M. K., and Patt, Y. N. Accelerating critical section execution with asymmetric multi-core architectures. *SIGPLAN Not. 44*, 3 (Mar. 2009), pp. 253–264.

[43] Winter, J. 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, pp. 29–40.

[44] Woo, S. C., Ohara, M., Torrie, E., Singh, J. P., and Gupta, A. The splash-2 programs: Characterization and methodological considerations. In *ACM SIGARCH Computer Architecture News* (1995), vol. 23, ACM, pp. 24–36.

[45] Wu, Q., Martonosi, M., Clark, D. W., Reddi, V. J., Connors, D., Wu, Y., Lee, J., and Brooks, D. A dynamic compilation framework for controlling microprocessor energy and performance. In *Proceedings of the 38th annual IEEE/ACM International Symposium on Microarchitecture* (2005), IEEE Computer Society, pp. 271–282.