Architecture and Circuit Cross-Cutting Approaches for Power-Efficient Multi-Core Processors

By

Hamid Reza Ghasemi

A dissertation submitted in partial fulfillment of

the requirements for the degree of

Doctor of Philosophy

(Computer Sciences)

at the

UNIVERSITY OF WISCONSIN-MADISON

2015

Date of final oral examination: 03/26/2015.

The dissertation is approved by the following members of the Final Oral Committee:
Nam Sung Kim, Associate Professor, Electrical and Computer Engineering and Computer Sciences
Mark D. Hill, Professor, Computer Sciences
Mikko H. Lipasti, Professor, Electrical and Computer Engineering
Karu Sankaralingam, Associate Professor, Computer Sciences
David A. Wood, Professor, Computer Sciences

Abstract

With technology scaling, which allows manufacturers to integrate more cores per chip, the performance of multi-core processors has been increased for the past decade. However, such a trend cannot be sustained because the power reduction per core has slowed down, while the maximum power consumption allowed per chip has not increased. This limits the maximum number of cores per chip. Therefore, improving the power efficiency of the existing and future multi-core processors becomes crucial for higher performance.

Traditionally, dynamic voltage and frequency scaling (DVFS) has been the most powerful technique to enable power-efficient multi-core processing. However, with aggressive technology scaling, the efficiency of DVFS has been continuously decreased due to the shrinking voltage scaling window. Prior efforts to improve the efficiency of DVFS often have not been (cost) effective. Besides, most of these solutions may be sub-optimal since they investigate only one abstraction layer: circuit or micro-architecture. Although solutions focusing on one abstraction layer reduce the complexity of the design process, they may increase inefficiencies in design by not exploiting the shared goals, limitations, and requirements across multiple design layers.

This dissertation discusses existing challenges to improve the power efficiency of multi-core processors, and proposes circuit and architecture cross-cutting techniques to further improve power efficiency of future multi-core processors. This dissertation identifies various challenges involved in the efficiency of DVFS and proposes solutions to overcome these challenges. In addition, it proposes

power efficiency techniques that are independent from and/or orthogonal to DVFS. In this dissertation, I make four main contributions:

First, I demonstrate a cost-effective power delivery technique to support per-core voltage domains for power-constrained multi-core processors. I exploit a very low-cost voltage regulator (VR) that uses the existing on-chip per-core power-gating (PCPG) devices available in most commercial processors and optimize a DVFS algorithm to efficiently work with this special VR. This technique considerably reduces the cost of VRs while it provides highly power-efficient per-core DVFS.

Second, I propose a last-level cache (LLC) architecture comprised of heterogeneous cell sizes to deliver both high-performance and low minimum operating voltage (VDDMIN). For low-power operation, the proposed LLC exclusively uses large cells that exhibit low failure rates at low voltages, enabling it to operate at low VDDMIN. For high-performance operation, it operates at a high enough voltage at which the failure rate of even small cells of the LLC is sufficiently low, providing the needed large LLC capacity.

Third, I propose a new technique to dynamically scale the number of cores and the amount of resources of processors for increasing power efficiency when DVFS technique becomes not a viable option for future processors. This technique is a DVFS alternative that can trade performance with power consumption through micro-architectural resource scaling mechanism (instead of voltage/frequency) to maximize performance under a power constraint. It also proposes a runtime policy that can effectively manage and leverage the proposed architectural mechanism to maximize performance under a power constraint.

Finally, I present a technique to dynamically adjust voltage regulators (VRs) to improve power efficiency. The VRs are the most critical component of processor power delivery sub-system and dissipates high power that is directly proportional to the power consumed by the processor. Traditionally, when a processor is in active state, all phases of VR will be active regardless of power consumption of processor. This leads to poor VR efficiency. This technique scales up and down the number of VR's active phases, by predicting the load current of processors, to increase the VR efficiency and reduce the VR's dissipated power.

With these mechanisms and analysis, this dissertation demonstrates multiple architecture/circuit techniques to improve power efficiency of multi-core processors when traditional DVFS technique is not efficient or not a viable solution.

Acknowledgements

During the graduate school, I have received support from many people. When I started writing the acknowledgement, I found it difficult to name all these people. I would like to apologize in advance if I leave out any deserving individuals.

First, I give thanks to Professor Nam Sung Kim. He contributed in all aspects of this dissertation. He spent many hours providing valuable feedback and discussions to clarify ideas, and ensuring our progress in each step of these research projects. He contributed highly in projects presented in this dissertation. I also would like to thank my dissertation committee members, Professor Mark D. Hill, Professor Mikko Lipasti, Professor Karu Sankaralingam, and Professor David A. Wood for their insights and guidance. In addition, I wish to thank Dr. Michel Schulte, Dr. Ken Vu, Dr. Srini Ramani, Dr. Alaa Alameldin, Professor Stark Draper, and Professor Ulya Karpuzcu for their valuable contributions and feedbacks.

I would like to thank my former and current colleagues at UW-Madison, including Jungseob Lee, Abhishek Sinkar, Arsalan Zulfigar, Syed Gilani, David Palfrom, Paula Aguilera, Philip Garcia, Daniel Chang, Mike Marty, Marc De Kruijf, Jayaram Boba, Dan Gibson, Yasko Eckhert, Arka Basu, Srinath Sridhara, Rathijit Sen, Gagan Gupta, Mohammad Shoaib Bin Altaf, Nilay Vaish, Tony Nowotski, Jayneel Gandhi, Newsha Ardalani, Jason Power, Marc Orr, Lina Olson, Joel Hestness, and Chris Feilbach. They have provided valuable feedback and discussions during graduate school. I also wish to thank my colleagues at AMD for their support.

Last but not least, none of these was possible without tremendous support and love from my family. I express my gratitude to my wife, Somayeh Sardashti. Her unconditional love and constant support helped me to be strong and positive. I also give my thanks to my parents. They have always supported and encouraged me in achieving my goals. Their love motivated me to stay focus and move forward. I also would like to thank my parents-in-law for their encouragements and support during the graduate school. In addition, I wish to thank my brothers, my brothers-in-laws, and my sister-in-laws for their encouragements.

I would also like to thank those who provided the financial support for my graduate studies. I have been supported by generous grants from AMD, IBM, the Wisconsin Alumni Research Foundation, and the National Science Foundation NSF (CCF-0953603, CCF-1016262, and CCF-095360).

Finally, this dissertation, for what it is worth, is dedicated to my wife, Somayeh Sardashti, and my parents, Amir Ghasemi and Kobra Ghaseminia.

Table of Contents

Chapter 1

Introduction

For the past several decades, the technology scaling has enabled manufacturers to integrate more cores per chip and increase the performance of multi-core processors. However, this trend cannot be sustained because the power reduction per core has slowed down while the maximum power consumption per chip has not increased. This limits the maximum number of cores per chip. Therefore, improving the power efficiency of existing and future multi-core processors is crucial to further increase their performance.

1.1 Power Management Techniques

Several power management techniques to improve the power efficiency of processors have been proposed. In general, the prerequisite of implementing these techniques is to have some level of scalability in a processor and then dynamically adapt the processor configuration to execute different phases of applications more efficiently. Therefore, processors require operating under different amounts of power envelopes (e.g., by adjusting voltage/frequency level) as well as different amounts of active cores and core resources (e.g., by adjusting the size of caches, the number of execution units, the issue width, and the size of the instruction queue and re-order buffer). I categorize the existing techniques into two general classes: techniques based on dynamic voltage and frequency scaling (DVFS) and DVFS alternatives (or non-DVFS techniques), as summarized in Table 1-1.

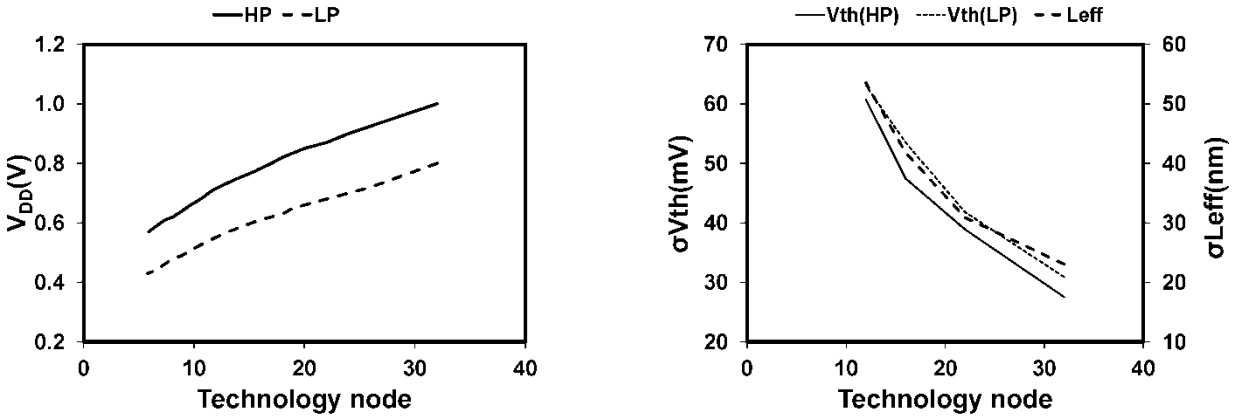


Figure 1-1: (a) Nominal voltage scaling and (b) process variation of the threshold voltage (Vth) and effective channel length (Leff) of the minimum size device with technology scaling. The standard deviation of the Vth values is the average of the NMOS and PMOS devices.

DVFS has been the most powerful technique to improve the power efficiency of processors. DVFS can improve power efficiency for various purposes: maximizing performance under a power constraint, minimizing power consumption under a performance constraint, or reducing the amount of heat generated by a chip. DVFS can achieve these goals by increasing or decreasing the voltage/frequency level of processors. However, the efficiency of DVFS has decreased in each technology generation as the voltage scaling window has reduced. As depicted in Figure 1-1(a), the nominal voltage of transistors has decreased with technology scaling. On the other hand, process variability, one of the main sources preventing on-chip caches comprised of static random access memory (SRAM) cells (and thus cores) from operating at low voltage, has considerably increased with technology scaling, as shown in Figure 1-1(b). At lower voltages and in the presence of process variation, the failure probability of individual memory cells in large-scale structures, such as caches, increases. This leads to more defective memory cells that fail to maintain their correct states. Several techniques have been proposed to deal with the reliability issues of SRAM cells and large-scale memory structures to

enable operation at low voltages. However, most of these techniques will considerably increase the chip area or significantly diminish the power reduction benefit of DVFS.

The efficiency of DVFS can be varied depending on **power delivery sub-system**, **voltage/frequency (V/F) scaling range**, and **runtime V/F assignment algorithm**, as shown in Table 1-1. In multi-core processors that power delivery sub-system supports one voltage domain for cores (i.e., chip-wide DVFS), the efficiency of DVFS can be limited because the chip-wide DVFS cannot exploit the different levels of power/performance trade-offs between cores that run different threads or programs. On the other hand, providing multiple voltage domains (i.e., per-core DVFS) requires multiple voltage regulators (VRs). A VR is required to efficiently deliver stable voltage and large current for a processor for efficient processor power management. Having multiple voltage domains and delivering power using multiple off-chip (or on-chip) VRs incurs a high cost for the platform and package designs or increases the chip area. In addition, the V/F scaling range affects the benefits of DVFS because it provides limited power window for executing applications. Therefore, the techniques that help to operate processors at lower voltages can increase this V/F range and improve the efficiency of DVFS. Fur-

Table 1-1: Taxonomy of power management mechanisms and my contributions

Power management techniques and their efficiency factors	DVFS	Power Delivery Sub-system	VR-Scale (chapter 6), LDO (chapter 3)
		Scaling Knob: V/F	Last-level cache with heterogeneous cell size (chapter 4)
		V/F Assignment Policy	LDO (chapter 3)
	DVFS alternatives	Power Delivery Sub-system	VR-Scale (chapter 6)
		Scaling Knob: processor resources	DRCS (chapter 5)
		Scaling Policy	VR-Scale (chapter 6), DRCS (chapter 5)

thermore, the DVFS policy or algorithm, which determines the V/F state at runtime, affects the efficiency of DVFS because it can find one sub-optimal V/F state for executing applications on the processor for each interval. For example, in a system with a single chip-wide voltage domain for cores, the algorithm will select one voltage for all the cores but the cores may run different threads or applications and they need different level of voltage and frequency to improve the power efficiency.

DVFS will not be a viable option or will be too expensive to support in the near future in spite of the efforts to improve the efficiency of DVFS. Technology scaling will practically stop the further reduction of the minimum operating voltage after a certain technology generation. Consequently, the voltage scaling window may vanish. Architects must pursue other research directions to offer alternative techniques to DVFS.

To implement the DVFS alternative techniques, I formulate the requirements and impacts into three attributes **power delivery sub-system**, **scaling knobs**, and **scaling policy**, as summarized in Table 1-1. Improving the efficiency of a power delivery sub-system can also improve the power efficiency of the entire system, even in the absence of DVFS techniques. One of the most critical components to deliver system power is off-chip VRs. These VRs supply the necessary power for various platform components, such as processors, chipsets, dynamic random access memory (DRAM) modules and storage devices, and they consume a large fraction of the total platform power. In addition, similar to a V/F knob in DVFS techniques, scaling knobs in DVFS alternative power efficiency techniques (e.g., core and resource scaling) may affect the benefits of processor power efficiency because they can provide diverse processor configurations, each with different power/performance trade-offs. Most DVFS alternatives provide micro-architectural mechanisms to achieve higher power efficiency. These micro-

architectural mechanisms may increase the design complexity. In addition, they may have a very high overhead for switching from one processor configuration to another. Furthermore, the scaling policy, which dynamically identifies the optimal processor configuration at runtime, may affect the power efficiency by failing to find the optimal processor configuration, having a very large search space, or having very limited search space.

1.2 Dissertation Contributions

In this dissertation, I discuss existing challenges to improving the power efficiency of multi-core processors and propose circuit and architecture cross-cutting techniques to further improve the power efficiency of future multi-core processors.

The traditional methodology for designing processors can potentially incur power efficiency of processors. A processor design is comprised of several abstraction layers (e.g., architecture, micro-architecture, circuit, and device). Each abstraction layer has its own goals, limits, and requirements and is often designed and implemented independently. Although employing such abstractions reduce the complexity of the design process, it may increase inefficiencies in design because we cannot fully exploit the potential benefits of recognizing the shared goals, limitations, and requirements across several design layers. Thus, jointly considering architecture, circuit, and technology challenges for future multi-core processors can provide more opportunities to make a processor more power-efficient. I pursue various techniques that cut across architecture and circuit designs to maximize the power efficiency of multi-core processors.

Table 1-1 summarizes the contributions of this dissertation. I make contributions to both DVFS techniques and DVFS alternatives for power management. I propose two techniques to improve the efficiency of DVFS: a last-level cache with heterogeneous SRAM cell sizes [1] and a cost-effective per-core DVFS [2]. I also propose an alternative to DVFS: dynamic resource and core scaling (DRCS) [3]. DRCS dynamically scales the number of active cores and processor resources during various phases of applications to improve processor power efficiency. Finally, I explore different techniques to scale VRs and increase their efficiency, resulting in higher processor power efficiency.

1.2.1 Cost-Effective Power Delivery Technique to Support Per-Core Voltage Domain

This research improves processor power efficiency by providing a cost-effective power delivery technique to support per-core voltage domains for power-constrained multi-core processors. Most commercial processors only have one chip-wide voltage domain for DVFS because splitting the voltage domain into per-core voltage domains and powering them with multiple off-chip voltage regulators incur a high cost for the platform and package designs. Although using on-chip switching VRs can be an alternative solution, integrating high-quality inductors and cores on the same chip has been a critical technical challenge. However, through architectural simulation studies, I observe that core-to-core voltage variations are relatively small for most DVFS intervals when the V/F of the cores executing a multi-thread workload are optimized to maximize the performance under a power constraint. Moreover, per-core power-gating (PCPG) devices augmented with some extra circuits can serve as low-cost VRs that can provide high efficiency when the core-to-core voltage variations are small.

In this research, I exploit a very low-cost VR that uses the existing on-chip PCPG devices available in most commercial processors and optimize a DVFS algorithm to efficiently work with this

special VR. The proposed technique considerably reduces the cost of VRs because the VR shares its most expensive component with the PCPG device while providing power efficiency as high as the state-of-the-art on-chip switching VRs. Furthermore, I propose a DVFS algorithm that is optimized for the proposed VRs by choosing core voltage levels to maximize their efficiency. The proposed DVFS algorithm considers the limited voltage difference between output voltage values (V_O) and the input voltage values (V_I) of the VRs to support highly efficient power delivery. In order to minimize the potential negative impact of the limited voltage range of the VRs in the proposal, the DVFS algorithm leverages (i) within-die (WID) and core-to-core (C2C) process variations that can lead to various frequency and power consumption trade-offs between cores even at the same operating voltages and (ii) thread migration at runtime to satisfy the performance requirement of the running threads.

1.2.2 Low-Voltage On-Chip Cache Architecture using Heterogeneous Cell Sizes

This technique explores multi-core processor architecture to provide the low minimum operating voltage cost-effectively. Although DVFS has been one of the most successful power reduction techniques, it is limited to some minimum operating voltage (i.e., VDDMIN). Therefore, reducing the VDDMIN will help to provide a larger voltage scaling window for DVFS, which can improve power efficiency. A large memory structure, such as last-level cache (LLC), in processors often determines the VDDMIN of the processor due to the impact of process variation on the reliability of SRAM cells at low voltages. Larger SRAM cells, which are less sensitive to process variability, support lower VDDMIN. However, large memory structures cannot afford to use such large SRAM cells due to the die cost. I made two main observations based on the simulations. First, the high-performance processor still spends a substantial fraction of its runtime in high voltage/frequency (V/F) states when the load

level of the processor is high. Second, the negative performance impact of having a smaller LLC capacity is reduced when processors are running at lower V/F states.

Motivated by these two observations, I propose an LLC architecture comprised of heterogeneous cell sizes designed for high-performance multi-core processors. This exploits (i) the DVFS characteristics of workloads running on high-performance processors, (ii) the trade-off between SRAM cell size and VDDMIN, and (iii) the fact that the latency between off-chip memory and on-chip core at lower V/F states is reduced. I exploit these characteristics to deliver both high-performance and low VDDMIN by architecting an LLC consisting of a spectrum of cell sizes. For low power operation, the proposed LLC exclusively uses large cells that exhibit low failure rates at low voltages so the LLC can operate at a low VDDMIN. On the other hand, for high-performance operation, it operates at a high enough voltage that the failure rate of even small cells of the LLC is sufficiently low for their use, providing the needed LLC capacity. At lower operating voltages, I disable subsets (e.g., ways in a set-associative LLC) of cells in order of size, starting with the smallest. Because at lower voltages a processor must run at lower frequencies, the frequency gap between on-chip cores and off-chip memory decreases. The result is that the performance penalty of having a smaller LLC in low V/F states is much smaller than it would be in high V/F states.In this way, my proposal provides low VDDMIN at small area, while a conventional LLC needs to use large cells across the entire cache to provide the same low VDDMIN. Using large cells will either require a larger die area for the same LLC capacity or result in a smaller LLC capacity for the same die area.

1.2.3 DRCS: Dynamic Resource and Core Scaling

In this dissertation, I also explore architectural mechanisms to maximize performance under a power constraint. By analyzing a wide range of workloads, I make two main observations that motivate my proposal. First, at a fixed frequency, each application (and each phase of application) exhibits a unique optimal processor configuration (i.e., the amount of resources, such as the size of on-chip memory and the number of execution units, per core and the number of operating cores). For example, for an application with high thread-level parallelism (TLP) but low instruction-level parallelism (ILP), using more cores with less resources per core can lead to higher performance than using fewer cores with more resources per core and vice versa. Second, the tradeoff between performance and power for each resource in cores is different.

Inspired by these observations, I propose a dynamic resource and core scaling (DRCS) technique that dynamically trades the performance with power consumption by jointly scaling the resources of cores and the number of operating cores. DRCS provides different processor configurations by trading the amount of resources per core with the number of operating cores. In order to dynamically find the optimal configuration for each application, I also propose an algorithm that monitors applications' runtime behavior and determines the best resource and core scaling configuration to maximize performance under a power constraint.

1.2.4 VR-Scale: Runtime Dynamic Phase Scaling of Processor Voltage Regulators

A voltage regulator (VR) is the backbone of power delivery sub-system. A VR dissipates high power that is directly proportional to the power consumed by the processor. A VR supplies current at a stable voltage (i.e., within an allowed margin) in the presence of rapid changes in current demand. It

also support fast, accurate and fine-grained voltage changes for efficient processor power management. These requirements become more stringent as technology scaling decreases processors' operating voltage, making it costlier to manufacture an efficient VR. Therefore, manufacturers have offered some knobs so that a processor can adapt a VR's operating parameters to cost-effectively satisfy the requirements with high efficiency.

In this study, I demonstrate that: (i) VR efficiency heavily depends on load current (i.e., current delivered to a processor) and a VR operating parameter (e.g., the number of active phases) at a given voltage; (ii) a processor running a parallel application mostly consumes small current due to aggressive power management; and (iii) when the processor is in an active state, all the VR components are always activated in the platform. (ii) and (iii) in turn lead to poor VR efficiency during most runtime. Second, I present VR-Scale, which dynamically scales the number of active phases based on the predicted load current for the next interval. My evaluations based on two Intel® processors running emerging parallel applications show that VR-Scale can reduce the total power consumed by both a processor and its VR with negligible performance impact.

1.3 Dissertation Organization

The rest of this dissertation is organized as follows. Chapter 2 discusses the background of power management techniques and related work. The next four chapters discuss this dissertation's contributions in detail. Chapter 3 explains a cost-effective per-core power delivery and optimized DVFS algorithm to support per-core DVFS. Chapter 4 describes the LLC design using heterogeneous SRAM cell sizes that provides low power operating voltage while supporting high-performance. Chapter 5

explains the DRCS technique as an alternative to DVFS. Chapter 6 explains the VR-Scale technique. Finally, Chapter 7 concludes the dissertation.

Chapter 2

Power Management Techniques: Background and Related Work

Under contemporary technology scaling, the efficiency of dynamic voltage and frequency scaling (DVFS) to enable power-efficient computing is decreasing due to the diminishing dynamic voltage (V) and frequency (F) range. Several proposals have been introduced to mimic or increase DVFS's utility for sustainable power efficiency and several architecture and micro-architectural techniques have been put forth as alternatives to DVFS. In the following chapter, I give an overview of power management techniques. I discuss most of these techniques, and explain how they affect processor power efficiency. I begin by revisiting the DVFS technique and its relevant background. I then discuss the proposals that improve the efficiency of DVFS. Finally, I discuss background and prior work related to DVFS alternative techniques.

2.1 DVFS

DVFS has been a widely used power management technique that changes supply voltage and clock frequency to improve processor power efficiency. DVFS has been exploited by diverse commercial processors across all computing segments, from the embedded and mobile market up to the server market. DVFS lowers supply voltage and clock frequency when the processor does not need high-

performance. In this way, it can improve processor power efficiency by reducing both dynamic and static power.

The dynamic power consumption of a processor is determined by $P_{dynamic} = \alpha \times C \times V^2 \times F$, where V is the supply voltage, F is the clock frequency, α is the switching activity level, and *C* is the capacitance of the processor circuit. The DVFS technique allows a processor to reduce both the *V* and *F* during processor execution, which leads to lower dynamic power. Dynamic power consumption is proportional to F^3 because clock frequency (*F*) is roughly proportional to supply voltage (*V*). Thus, as DVFS mechanism decreases the processor V/F level, it reduces dynamic power consumption cubically. In addition to lowering dynamic power, DVFS can reduce static power consumption. Static power, including gate leakage power and subthreshold leakage power, is affected by *V*. Therefore, reducing *V* affects both subthreshold and gate leakage. DVFS influences both dynamic and static power, which explains why DVFS has been very popular for power efficiency.

Several prior studies have investigated the benefits of DVFS for multi-core processors and have proposed techniques to advance its efficiency. In this dissertation, my focus is on improving the efficiency of DVFS by providing cost-effective per-core DVFS and lowering the minimum operating voltage using heterogeneous static random access memory (SRAM) cell sizes. I therefore present work related to these topics in detail. I classify attributes that affect the efficiency of DVFS into three categories: **power delivery sub-system**, **V/F range**, and **V/F assignment algorithm**, as shown in Table 2-1. For each discussed related work, I explain how it improves the efficiency of DVFS and which factors it enhances.

2.1.1 Voltage Regulator: Basic Block for DVFS

In a processor platform, one of the most critical components is voltage regulators (VRs). VRs supply necessary power for various platform components such as processors, chipsets, dynamic random access memory (DRAM) modules and storage devices. The key role of a VR is to convert high voltage level (e.g. 5V) to lower voltage levels that is operational (e.g. 1.2V ~ 0.6V) for processors. VRs are responsible to deliver large current while providing a constant voltage level.

A VR uses a feedback control loop to provide a stable output voltage. DVFS algorithm assigns a required voltage level as reference voltage (Vref) to VR. It compares the actual output voltage (Vo) to the Vref. Any difference between Vo and Vref is amplified and used to control the regulation element to decrease the voltage error. This forms a negative feedback control loop to fix the output voltage. Whenever the load current of the processor changes, the output voltage of VR (i.e., the input voltage of processor) may have a voltage droop. It creates an error in the VR feedback. So, the voltage regulator will use the negative feedback to reduce the error to zero and fix the output voltage again.

Multi-phase VR: one of the technique to improve the efficiency of VR is to use multi-phase VR design. Today's VR are designed with 6~8 VR phases. For example, a VR with six phases comprises of six identical circuits working together to supply the required power of the CPU. However, these phases are not working at the same time. Whenever one VR phase is "ON", all other phases are resting. In this way, each phase of VR will be "ON" 1/6 of the time in a VR with 6 phases. Having multiple phases in VR helps the VR to provide more stable output voltage, which lowers the voltage noise levels. Another benefit of having multiple phases is that each phase will have more time to rest resulting in lower operating temperature and higher life-span of VR.

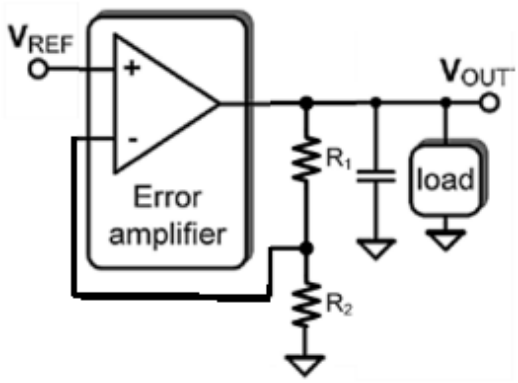


Figure 2-1: a simple diagram for LDO Linear drop-out VR.

Linear drop-out (LDO) VR: An LDO regulator is a linear VR that can provide output voltages (i.e., the input voltage to the processors) even very close to the supply voltage level. Figure 2-1 shows the simple diagram of the LDO VR. The main components of LDO VR are a power field-effect transistor (FET) and a differential amplifier (error amplifier). One input of the differential amplifier monitors the fraction of the output determined by the resistor ratio of R1 and R2. The second input to the differential amplifier is from a stable voltage reference (Vref). If the output voltage rises too high relative to the reference voltage, the drive to the power FET changes to maintain a constant output voltage. The output voltage is calculated by $V_{out} = (1 + R_1 / R_2) \times V_{ref}$.

Table 2-1: Taxonomy of power management mechanisms and DVFS related work

Name	Power Delivery Sub-system	Scaling Knob	Scaling Policy
Li *et al.* [142]	Off-chip VR	V/F + Core	-
Kim *et al.* [4]	On-chip Switching VR	Per core V/F	-
Eyerman *et al.* [8]	On-chip Switching VR	Per core V/F	Fine-grained micro-architecture driven
Teodorescu *et al.* [5]	-	V/F + WID process variation	Linear programing: trading performance versus power
Rangan *et al.* [11]	Fixed VR output but different voltage level	Per-core V/F	Thread migration
6-transistor (6T) SRAM cell [12] [73] [14]	-	Increasing V/F range	-
Separate voltage do-main [25]	Separate voltage domain for caches	Increasing V/F range	-
8T, 10T or ST SRAM cell [15] [16] [18] [19]	-	Increasing V/F range	-
Logic technique for SRAM cells [20] [14]	-	Increasing V/F range	-
Architecture technique [21] [22] [23] [24]	-	Increasing V/F range	-

2.1.2 DVFS and Per-Core Voltage Domain

Although a diverse range of commercial processors support DVFS, most of them have only a single chip-wide voltage domain due to the high cost of supporting multiple voltage domains. As more cores share the chip-wide voltage domain, the performance/power benefit of DVFS diminishes. This is because a single voltage domain cannot effectively exploit runtime performance (i.e., IPC) variations across cores running multiple threads or different applications. For example, cores running memory-intensive threads can operate at a lower V/F level without impacting the performance. While cores executing compute-intensive threads must operate at a higher V/F level to maximize performance.

Consequently, many researchers have investigated various DVFS algorithms to exploit multiple voltage domains effectively [4, 5].

Table 2-1 categorizes several important DVFS techniques. Li *et al.* [6] analyze the performance of DVFS combined with dynamic core scaling for multi-core processors for parallel workloads. They exploit the observation that parallel workloads with a limited problem size do use all cores efficiently. Thus, they jointly adjust the number of active cores along with performing DVFS to maximize performance under a power constraint. Kim *et al.* [7] demonstrate the potential benefit of per-core DVFS using on-chip switching VRs for embedded processors and provide detailed background on switching VRs using air-core inductors. S. Eyerman *et al.* [8] also evaluate the benefits of fine-grained applications of DVFS and propose a fine-grained micro-architecture-driven DVFS mechanism.

Many researchers have also studied the impact of within-die (WID) process variations in multi-core processors. WID variations can lead to core-to-core (C2C) frequency and power variations [9, 5, 10]. They propose a DVFS algorithm that exploits the C2C frequency and power variations. Teodorescu *et al.* investigate a DVFS algorithm based on linear programming to maximize the performance of multi-core processors under a power constraint [5]. Their DVFS algorithm also exploits WID C2C frequency and power variations for workload scheduling and power management.

Rangan *et al.* propose a thread migration technique to minimize the cost of the transition time for the VR output voltage [11]. They introduce voltage domains in which each core operates at a fixed but different voltage level. If threads require different V/F levels for power-efficient operations, they migrate to the cores that can provide an appropriate performance level instead of changing the V/F of the cores.

2.1.3 Coping with Minimum Operating Voltage

The ever-increasing process variability in nanoscale technology, such as random dopant fluctuation (RDF) and line edge roughness (LER), limits voltage scaling of on-chip memory elements to some minimum operating voltage, VDDMIN. At lower voltages, multi-core processors cannot operate reliably due to defective memory cells that fail to maintain their correct states. As a result, the failure probability of individual memory cells in large-scale structures such as last-level caches (LLCs) often determines the VDDMIN for the processor.

The simplest approach for lowering the VDDMIN for a target yield is to increase the size of the transistors in the SRAM cells. This makes the transistors less sensitive to mismatches induced by process variations, which are the cause of the higher cell failure rates at low voltages. However, in current state-of-the-art multi-core processors, LLCs already occupy more than 50% of total die area. Furthermore, the capacity of such LLCs needs to be increased as more cores are integrated into a single die. This makes it impractical to substantially increase the size of individual SRAM cells.

Table 2-1 also categorizes several important techniques for coping with minimum operating voltages. Many circuit-level techniques have been proposed to deliver robust conventional 6-transistor SRAM cells at low voltage for large on-chip caches [12] [13] [14]. These techniques use special circuits and often modulate supply, body, word-line, and bit-line voltages applied to SRAM cells to make the SRAM cells more robust at a low voltage. Such techniques often incur an additional cost for the specialized circuits, which influences design, validation, and testing complexity, although they can reduce the failure probability of SRAM cells by an order of magnitude. Alternatively, other SRAM circuit designs, such as 8T, 10T, and ST SRAM cells, have been adopted to lower the VDDMIN [15]

[16] [17] [18] [19], though at the cost of substantial increases in SRAM cell area (e.g., 100% increase for the ST cells).

In another approach to making on-chip caches more defect-tolerant at low voltages, several logic-level techniques, such as redundancy and error-correction code (ECC), have been adopted [14] [20]. These techniques were originally introduced to improve the manufacturing yields of dynamic random access memory (DRAM) and SRAM arrays. A small number of redundant columns and/or rows of SRAM cells are made available to replace a column or row that contains a cell (or cells) with manufacturing defects such as stuck-at-faults. After a manufacturing test, SRAM arrays containing defective cells are reconfigured such that redundant columns/rows replace ones containing bad cells. Meanwhile, ECC was adopted to protect DRAM and SRAM arrays from particle-induced soft errors. However, because RDF and LER arise at low operating voltages, ECC is now also used to correct SRAM cell failures at low voltages.

In separate approaches, micro-architectural techniques have been proposed to cope with the V_{DDMIN} challenge [21] [22] [23] [24]. In effect, to lower the V_{DDMIN}, these approaches identify defective cells and disable the entire lines of the cache that contains those cells [21] [22]. However, they do not provide the available cache capacity deterministically because each die will exhibit a different number of failing cells at each voltage level. In addition, they often require a large number of post-manufacturing programming bits to identify cache lines that must be disabled. Although it is claimed that on-line testing schemes can be adopted to verify the integrity of each cell, it would be very challenging to test millions of cells repeatedly at different voltage levels and specific temperatures every time you turn on your processors; temperature also influences SRAM failure at low voltages. Thus, as the voltage is lowered, these approaches implement incrementally stronger ECC at the expense of a

reduced cache capacity. Up to half of the total cache capacity can be diverted to storing check bits [24]. To avoid the VDDMIN problem, these techniques use different voltage domains for processor cores and on-chip caches [25]. L1 caches are connected to a high supply voltage while processor cores can operate at a very low voltage. To reduce power consumption of the L1 caches connected to a high voltage, these techniques introduce a filter cache that operates at a very low voltage between L1 caches and processor cores, thereby connecting them to three separate voltage domains. As a result, other challenges associated with multiple voltage domains arise, e.g., design, verification, and testing complexity due to domain crossing and a higher cost of voltage regulators.

2.2 DVFS Alternatives

Although DVFS has been the most powerful technique for trading performance with the power consumption of processors, its efficiency has been reduced because the range of V/F scaling for commercial processors has been continuously decreased. Researchers have proposed several alternative technique for DVFS that mostly scale processor resources, using micro-architectural mechanisms, to the needs of running applications.

These micro-architectural mechansims exploit different **power delivery sub-systems**, **scaling knobs**, and **scaling policies**, as shown in Table 2-2. Each technique requires different power delivery sub-system support, such as the need to turn on/shut down specific resources in processors. Also, each technique uses different scaling knobs (such as cache resizing or ALU scaling). These micro-architectural mechanisms provide different processor configurations, which increases the design complexity. Furthermore, to select the optimum processor configuration, the system needs an algorithm or scaling policy to decide the best processor configuration in runtime.

Table 2-2: Taxonomy of power management mechanisms and DVFS alternatives

Name	Power Delivery Sub-system	Scaling Knob	Scaling Policy
Petrica *et al.* [34]	Shut down each lane separately	Four de-configurable lanes in each processor pipeline	Online sampling & training Heuristic online search
Watanabe *et al.* [31]	Shut down unused resources	Instruction engine Execution unit	-
Iyer *et al.* [26]	Shut down partial CAM and RAM	Effective pipeline width The size of RUU	Runtime profiling of 12 configurations
Albonesi *et al.* [27]	Shut down partial CAM and RAM	ROB, F-, I-REG, D-, I-cache, IIQ, FIQ, LSQ	Per-unit monitoring and reconfiguring
Lee *et al.* [28]	Shutdown all resources of any size	Pipeline depth & width, D-, I-, L2-cache, memory latency, BP, LSQ, Reg	Spatial sampling, providing regression model, genetic algorithm to find optimal configuration
Kontorinis *et al.* [29]	Shut down each segment of the resources	D-, I-, L1-cache, ROB, IIQ, FIQ, IREG, FREG, LSQ, FPU, ALU	Threshold-based mecha-nism for the number of conflicts per cycle for each configuration
Homayoun *et al.* [30]	-	Sharing ROB, PRF, IQ, LSQ, in 3D cores	When a resource is full, a thread asks for a new partition of the resource.
Khubaib *et al.* [33]	Shut down unused resources	Transform between OoO to multiple in-order	If processor runs more than two threads, it switches to in-order core mode

Several prior studies have investigated the benefits of adaptive resource scaling for reducing power consumption [26, 27, 28, 29]. These efforts focus on uniprocessors and do not capture the effects of spreading threads over more or fewer cores and scaling shared resources, such as LLC, on performance. Iyer et al. observe that the utilization of architectural components considerably changes over long runtime periods and explore an adaptive resource scaling technique to minimize average power consumption [26]. Albonesi et al. explore various hardware/software techniques to adapt resources of a uniprocessor for higher power efficiency [27]. Lee et al. examine a wide range of uniprocessor design spaces to analyze performance and power efficiency trends and the limits of adaptively reconfiguring the resources of uniprocessors [28]. Kontorinis et al. propose a table-driven adaptive resource scaling technique with a runtime algorithm, which reduces a considerable amount of

peak power consumption with little performance loss [29]. After determining the best configuration, the processor will run for a long period of time to amortize the reconfiguration overhead.

Homayoun et al. propose a configurable heterogeneous processor architecture exploiting 3D stacking technology [30]. The resources of different layers are shared at a fine granularity between vertically stacked cores. As a result, each core can borrow/lend the resources from/to other cores as needed by the thread running on a core. Watanabe et al. propose a flexible in-order uniprocessor architecture to cope with a shrinking range of V/F scaling [31]. The number of instruction engines (IEs) and execution units (EUs) is scaled up/down independently to maximize the performance of single-thread applications. However, they do not explore any runtime algorithm for dynamically provisioning shared resources, such as LLCs, and the number of operating cores. Gibson et al propose an architecture that can scale up (or down) its execution units and instruction window to improve single thread (or multi-thread) performance [32]. This architecture, called Forwardflow, dynamically extracts the data flow graph of instrcution sequence and uses this flow to guide processor frontend. Forwardflow organizes processor backend with separate units that can be (de-)allocated.

Khubaib et al. propose a micro-architecture that can transform its Out-of-Order (OoO) core into an in-order core by totally disabling resources required for OoO execution, but they do not propose or evaluate any runtime system to determine when to reconfigure it [33]. Petrica et al. propose an adaptive approach to fill the gap between fine-grained adaptive micro-architecture and a core-level power-gating technique [34]. They do not jointly scale the number of operating cores or shared resources such as LLC. Suleman et al. propose a feedback-driven approach to maximize performance by dynamically varying the number of threads at runtime [35]. They use fixed number of cores in their study, and a

special mechanism to dynamically change the number of operating cores at runtime. Moore and Childers also propose a dynamic approach that can choose the optimal number of threads based on offline-profiling and online-configuring methods [36]. Finally, some prior studies have investigated adaptive approaches based on machine learning, control theory, etc. for dynamic resource, core, and/or V/F scaling techniques for power and/or thermal management or performance improvement under a power constraint [35] [36] [37] [38] [39].

Chapter 3

Supporting Low-Cost Per-Core DVFS

A wide range of commercial processors, from smartphones to servers, have exploited dynamic voltage and frequency scaling (DVFS). Due to the high overheads of voltage regulators, current designs mostly support a single chip-wide voltage domain. However, increasing the number of cores that share a single chip-wide voltage domain would diminish the efficiency of DVFS. A single shared voltage domain does not allow the processor to effectively leverage the runtime performance differences across different cores. Thus, many researchers have investigated various techniqes to make use of multi-domain DVFS and improve the efficiency of DVFS .

The efficiency of DVFS depends on multiple factors including: power delivery sub-system and DVFS algorithm. Power delivery sub-system is responsible for supplying the power required to operate a processor and its components. The required power/performance level of each core can be different during runtime. To support per-core DVFS and improve the power efficiency of a processor, we need power delivery sub-systems that supply different levels of power envelopes for each core separately. Furthermore, a DVFS algorithm, which determines the optimum voltage/frequency (V/F) for each core, plays a critical role in improving the efficiency of DVFS.

My goal is to provide a solution to improve the efficiency of DVFS with minimum overhead. Thus, in this chapter, I propose a power delievery technique to support per-core voltage domains with very low overhead along with an algorithm optimized for this specific approach. The rest of the chapter is as

follows. Section 3.1 demonstrates the benefits of a per-core voltage domain. I show that high-performance multi-core processors using per-core DVFS can deliver considerably higher performance than using chip-wide DVFS under a power constraint. Section 3.2 discusses the challenges associated with supporting per-core voltage domains. Section 3.3 demonstrates that core-to-core (C2C) voltage variations are relatively small, even when the core voltages are optimized to maximize the performance of a power-constrained processor at each DVFS interval. In section 3.4, I propose a low-cost power delivery technique using a low-dropout (LDO) voltage regulator (VR) that exploits (i) small C2C voltage variations and (ii) on-chip per-core power-gating (PCPG) devices. Section 3.5 demonstratess that a processor using this proposed technique is as effective as a processor using on-chip per-core switching VRs with significantly lower area overhead. Finally, Section 3.6 summarizes the chapter.

3.1 Benefits of Per-Core Voltage Domains

To show the importance of supporting per-core DVFS, Figure 3-1 compares the efficiency of different DVFS techniques in an 8-core multi-core processor. Because this study targets server-class multi-core processors under a power constraint, I use $MIPS^3/W$ (Million instructions per second cubed per Watt) to emphasize the performance aspect of the processors more than MIPS/W (Million instructions per second per Watt) would [40]. For each application, I measure $MIPS^3/W$ when supporting a chip-wide V/F domain (chip-wide VR), per-core V/F domains (per-core VR), and per-core V/F domains exploiting within-die process variations (per-core VR + WID PV). All the results shown in Figure 3-1 are normalized to the 8-core processor with a chip-wide V/F domain without considering the power overhead of VRs. I use different classes of applications, including commercial

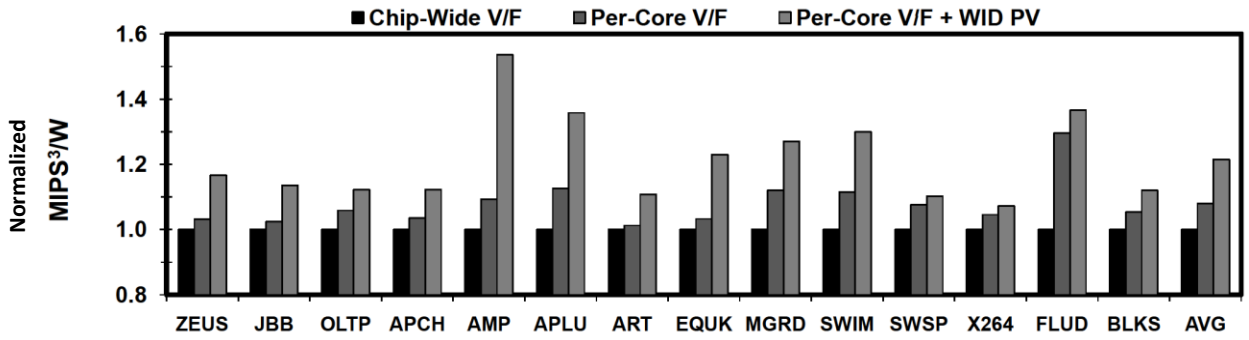


Figure 3-1: MIPS^3/W comparison of 8-core processors supporting a per-chip V/F domain, per-core V/F domains, and per-core V/F domains exploiting WID process variations. I apply an oracular approach similar to the one shown in [7] for each runtime DVFS interval. Each interval is comprised of 10 million executed instructions, which is equivalent to a few hundred microseconds, depending on IPC values. A total of 1 billion instructions is executed after 100 million instructions are executed for warming up the on-chip caches.

workloads, SPEC-OMP benchmarks and PARSEC benchmarks (see Section 3.5.2 for details on the applications).

For these experiments, I use an ideal DVFS algorithm. This algorithm determines DVFS state per execution interval. There are various methods to predict the optimum V/F state of each interval based on the execution history. To provide a fair and accurate comparison between the different DVFS techniques, independent of the algorithm in use, I assume that the optimum V/F state at every interval is known in advance. I employ oracular greedy optimization to extract the optimum V/F state per execution interval. To find the configuration of maximum power efficiency, the algortihm is optimized to consider MIPS^3/W as the metric. See Section 3.5.2 for the details of the DVFS algorithm, the processor configurations, and the methodology of modeling C2C frequency and power variations due to WID process variations.

Per-core DVFS (Per-core V/F) increases MIPS^3/W by an average of 8%. By exploiting the C2C variations, Per-Core V/F + WID PV can further improve MIPS^3/W by an average of 22% over chip-

wide DVFS. Furthermore, as I increase the number of cores per processor, I observe that per-core DVFS achieves even more of an improvement in $MIPS^3/W$ than chip-wide DVFS. For example, the $MIPS^3/W$ improvements of 12- and 16-core processors using per-core DVFS over chip-wide DVFS are nearly 3× and 4× higher than that of a 8-core processor, respectively. The increase becomes even larger when the C2C process variations are exploited. This signifies the growing importance of supporting per-core DVFS to maximize performance and power efficiency under a power constraint.

3.2 Challenges to Support Per-Core Voltage Domains

Supporting per-core voltage domains can allow multi-core processors to exploit C2C frequency and power variations, and hence significantly increase performance and power efficiency. However, most commercial processors have only one chip-wide V/F domain. This is because splitting the voltage domain and providing multiple off-chip VRs incurs a high cost for the platform and package designs. Figure 3-2 illustrates the impact of splitting the voltage domain to provide per-core DVFS on the overall VR capacity required. Assume that the maximum power consumption of the processor is limited to 120W and there are four cores. When all four cores are running, each core can consume up to 30W; thus, it seems that each per-core VR only needs to support up to 30W. However, for example, when only two out of four cores are active due to limited parallelism, the two active cores can run at higher V/F (e.g., Intel® Turbo Boost TechnologyTM [41]) without violating their thermal and maximum supply voltage constraints for reliability. If the two active cores consume 40W at such an operating V/F, the capacity of each VR needs to be increased to 40W and the total combined capacity of all the VRs becomes 160W.

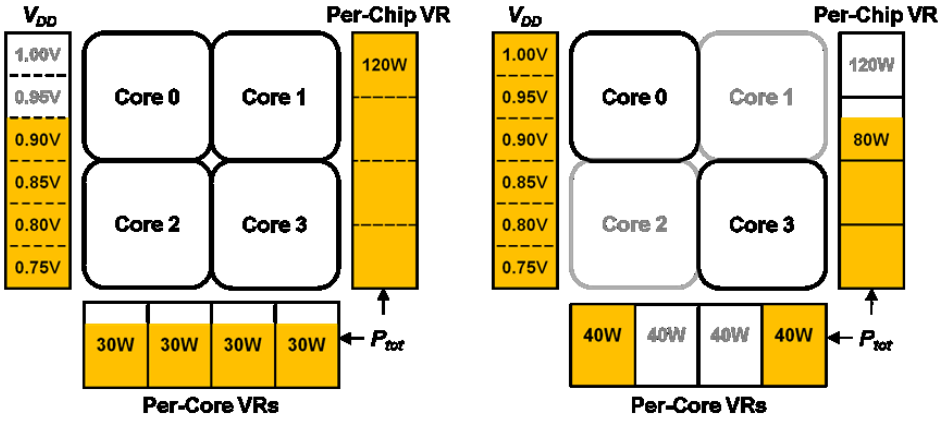


Figure 3-2: Impact of splitting the voltage domain on the overall VR capacity of to provide per-core voltage domains. All cores are active and consume a total of 120W in (a) and only two cores are active in (b).

When the voltage domain is shared, however, a 120W VR is still sufficient for such a case; the total power consumption of two cores running at the turbo mode is a total of 80W, which is below the maximum capacity of the VR. Although it is feasible for only a subset of cores to run in turbo mode, we cannot increase the VR capacity for only a subset of the cores. This is because cores are put into turbo mode in a round-robin fashion to prevent excessive aging of a specific core or subset of cores, requiring us to provide the capacity for turbo mode for all the cores. Finally, increasing WID process variations leads to substantial C2C frequency and (leakage) power variations [42]. Thus, the per-core VR capacity is determined by such cores, increasing the overall VR capacity even further.

The increased total power capacity requires larger components for VRs and more package pins for power delivery. Note that form-factor is critical even for server platforms to maintain high integration density in data centers. VRs are the second largest components next to dynamic random access memory (DRAM) modules. VRs occupy 63% more platform area than the CPU, the third largest components [43]. Furthermore, many commercial chips are heavily constrained by the available pins. Nearly a half of all pins are already dedicated for power delivery, and increasing VR capacity would

require even more pins. Although on-chip switching VRs can lower the platform and package costs associated with multiple off-chip VRs [44], integrating cores and high-quality inductors for VRs on the same chip has been also a major technical challenge for manufacturers, potentially impacting both the efficiency of the VRs and the yield of dies [45].

3.3 C2C Voltage Variations

Although the per-core DVFS algorithm is fully flexible to assign very wide range of voltage levels for each individual core, I observe that the voltage range exploited by DVFS algorithm per interval is in fact not very wide. This trend can be observed from collected data statistics of V/F assignment for each core per interval along with the experiment for Figure 3-1. In fact, I found that at each DVFS interval the maximum voltage difference between cores in a processor supported by per-core voltage domains is usually small. Figure 3-3 shows that the maximum voltage difference between cores for the Per-Core V/F is less than or equal to 100mV for at least 90% of the execution intervals in most applications. I also observe similar statistics for the Per-Core V/F + WID PV.

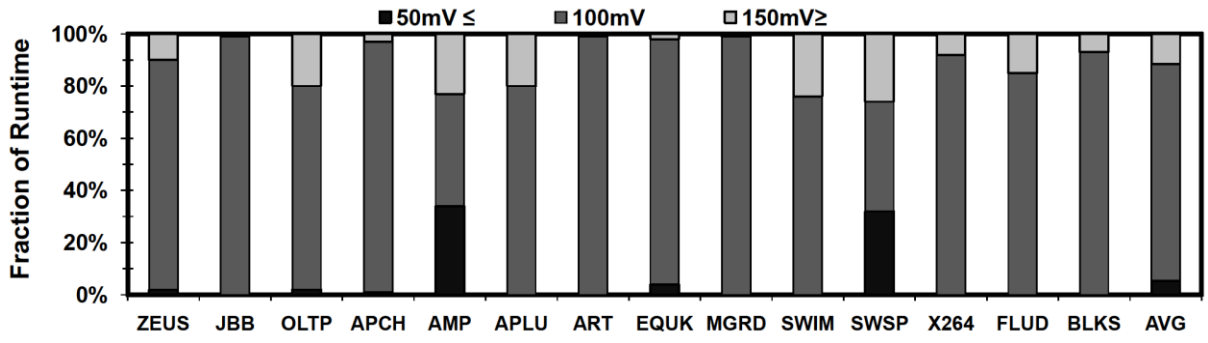


Figure 3-3: Fraction of intervals exhibiting various maximum voltage differences between cores for the “Per-Core V/F” case presented in Figure 3-1.

This observation makes a special type of voltage regulators a good candidate to be exploited as per-core VRs. This type of VRs is called low-dropout (LDO) and is derived from PCPG component in processors. LDO VRs can operate efficiently under some voltage conditions, which I will briefly discuss in the next section. Furthermore, an LDO VR can be implemented very cost-effectively since it does not require inductors or capacitors [46] and it can share its largest component (i.e., the output device) with a PCPG device.

3.4 PCPG-Based LDO VRs

PCPG devices are typically provided for commercial multi-core processors to reduce standby leakage power of idle cores [47]. In active state, a PCPG device incurs a slight voltage drop between the supply voltage and the actual voltage applied to the core. The voltage drop is inversely proportional to the size (i.e., total transistor width) of the PCPG device for a given amount of total current (dynamic + leakage) drawn by the core. In fact, the voltage applied to the core can be modulated by controlling the effective width (i.e., resistance) of the PCPG device [48].

As illustrated in Figure 3-4(a), a PCPG device has many parallel transistors and on/off signal buffers.It is similar to the largest component (i.e., the pass device between V_I and V_O) of a typical LDO VR. In other words, an LDO VR can be implemented by augmenting a PCPG device with a feedback control circuitry which is comprised of an error amplifier, an analog-to-digital converter, and a reference voltage generator. Hazucha et al. [46] reported that the output device and its buffers, both of which can be shared with a PCPG device, accounted for 83% of the total LDO VR area. Since a PCPG device consumes 5%-10% of a core's area [49], I estimate that the extra overhead due to the feedback

control circuitry to implement LDO VR is 0.85%-1.7% of the core's area. By contrast, on-chip switching VRs require large inductors and capacitors. As a result, a switching VR has nearly four times larger chip area than a comparable LDO VR [50]. Furthermore, LDO VRs can provide faster transient responses than switching VRs [51]. Unlike switching VRs, LDO VRs do not inject switching noise in the substrate, which is desirable for the operation of highly sensitive mixed signal circuits.

Figure 3-4(b) shows two different approaches to distribute supply voltages to an 8-core processor with per-core V/F domains. Both approaches use a first stage off-chip VR to convert 5V to an intermediate voltage level, V_I of on-chip per-core VRs; we cannot supply 5V for on-chip switching VRs directly considering the oxide-reliability of nanoscale transistors implementing both VRs and cores. This voltage is further down converted using on-chip per-core VRs to the voltage ($V_O[i]$) required by core *i*. The arrangement on the left uses LDO VRs (i.e., PCPG devices augmented with the control and reference circuitry to implement LDO VRs).

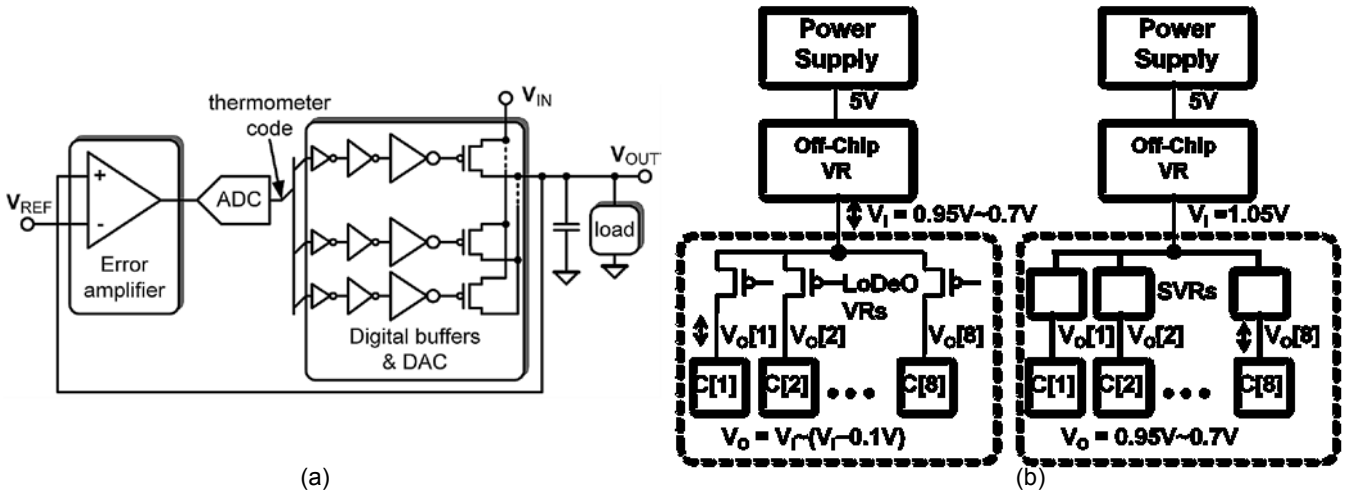


Figure 3-4: (a) A typical LDO VR architecture; the cartoon is reproduced from [46]. (b) An example of V_I and V_O ranges of LDO VRs in the left and switching VRs denoted by SVRs in the right for supporting per-core voltage domains; I ignore the default voltage drop of the LDO VRs due to the small resistance of the fully turned-on PCPG devices for the illustration purpose in the cartoon. "C[*i*]" in (b) denotes core *i*.

The efficiency of a LDO VR is a function of its V_O/V_I ratio. When the voltages demanded by individual cores are restricted to a limited range (e.g., within 100mV of one another as shown in Figure 3-3), a high VO/V_I ratio can be achieved for all the cores by adjusting the V_O of the first stage (i.e., V_I of the second stage) such that it is sufficient to provide the highest V_O demanded by any of cores. Thus, a processor adopting per-core LDO VRs can be tuned to achieve high efficiency by jointly optimizing both their V_I and V_O. The arrangement on the right uses per-core on-chip switching VRs to provide the necessary core voltage. A switching VR uses two active devices, inductors and capacitors to provide high voltage conversion efficiency across a wide range of V_O. This efficiency is primarily determined by the switching losses in the active devices and their conduction losses. The V_I value for switching VRs is fixed to 1.05V in this example.

In summary, an LDO VR can be implemented very cost-effectively since (i) it does not require inductors or capacitors [46] and (ii) it can share its largest component (i.e., the output device) with a PCPG device. Furthermore, its efficiency can be very high when cores running multiple threads or applications demand similar voltage values.

3.4.1 Efficiency Comparison: LDO versus Switching VRs

Figure 3-5 compares the efficiency of a switching VR with that of a LDO VR (the on-chip second stage only in (a) and both the off- and on-chip stages in (b), respectively). The efficiency of LDO VRs is higher than switching VRs when VI − VO is small (or VO/ VI is high). If VI − VO is more than

100mV, the efficiency of LDO VRs is usually lower than that of switching VRs as shown in Figure 3-5(a). I model the efficiencies of both switching and LDO VRs assuming each core consumes the maximum allowed current for each operating voltage. To measure the maximum efficiency of the switching VR at each operating point (i.e., voltage/current), I search for and activate the optimal number of phases out of eight available phases for a given voltage/current. Table 3-1 summarizes the key design parameters of various VR stages described in this study.

Off-chip switching VR designs, which are built with off-the-shelf components, typically have very high efficiencies (> 90%) due to low loss inductors and capacitors. The efficiency of off-chip switching VR is computed using approach presented by Klein et al [52] with V_I fixed to 5V. Their efficiency reaches a maximum for a certain load current and then drops with further increase in current due to an increase in conduction losses. Consequently, as the V_O of off-chip regulator for LDO VRs decreases, the efficiency degrades, and thus the overall efficiency of LDO VRs becomes slightly lower than that of switching VRs, as plotted in Figure 3-5 (b).

Table 3-1: Summary of VR design parameters.

	Off-chip Switching VR	**On-Chip Switching VR**	**On-chip LDO VR**
V_I/V_O	5V/1.05V to 5V/0.85V	1.05V/0.95V to 1.05/0.7V	0.95V/0.7V to 0.7V/0.95V
Technology	N/A	32nm	32nm
f_{sw}	300KHz	100MHz	N/A
L/phase	360nH (r_L = 0.5mΩ)	63.5nH (Q= 20 @100MHz)	N/A
No. of Phases	6	8	N/A

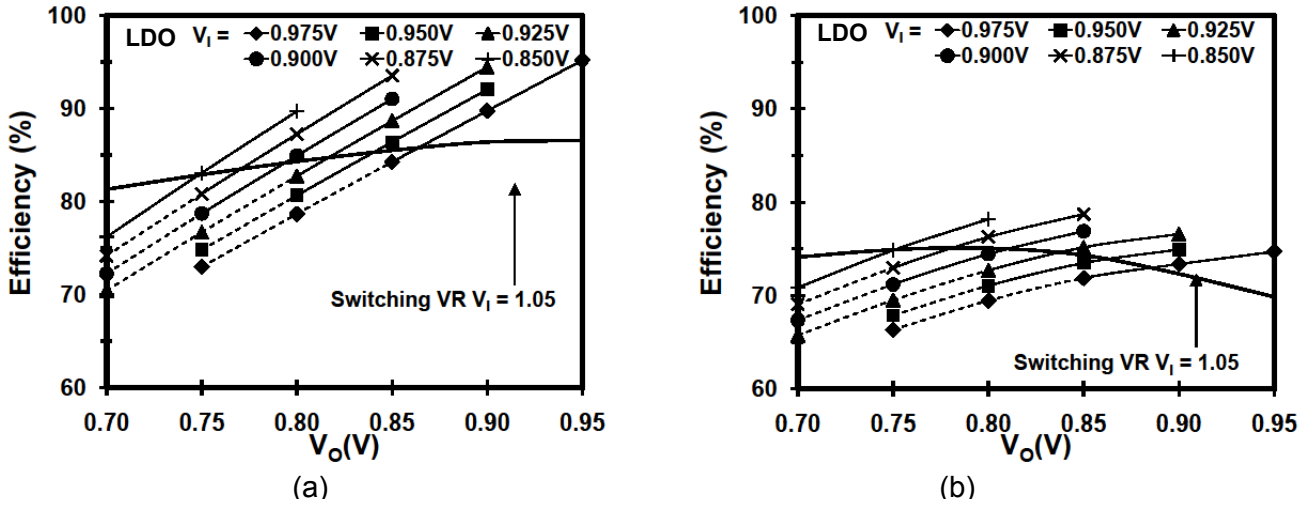


Figure 3-5: Efficiency comparison between switching and LDO VRs. The efficiency of (a) the on-chip (the second stage) only and (b) both the off- and on-chip (the first and second stages) are considered.

The efficiency of an LDO can be calculated as:

$$\eta_{LDO} = \frac{V_O \times I_O}{V_O \times I_O + (V_I - V_O) \times I_O + V_{bias} \times I_Q} \qquad (3\text{-}1)$$

where I_Q is the quiescent current of the LDO and V_{bias} is the biasing voltage for the reference and feedback control circuitry. A steady analog V_{bias} = 0.9V generated on chip from the variable V_I is assumed in this work. The current efficiency of a typical LDO VR is defined by [53]:

$$\eta_I = \frac{I_O}{I_O + I_Q} \qquad (3\text{-}2)$$

η_I is a measure of the power loss in the control and biasing circuitry of the LDO. On-chip LDO designs with current efficiencies in the range of 95% to 99% have been reported. The LDO efficiency is computed assuming a current efficiency of 97% [46] at I_O corresponding to 120W/0.9V.

Lee et al [54] modeled the efficiency of switching VRs with integrated inductors for different CMOS technology generations. The efficiency is mainly a function of the inductor *Q* factor. Inductors in CMOS processes are made from the available metal layers and attain low *Q* values for realistic dimensions due to the substrate losses and frequency dependent conduction losses. Lee et al [54] showed that a fully monolithic switching VR achieves ~62% efficiency with on-die inductors in 90nm CMOS. This low efficiency is not acceptable considering the performance benefit that can be brought by per-core voltage domains under a power constraint. The efficiency can be improved by using alternate inductor technologies with high *Q*. This may include inductors with magnetic materials compatible with a CMOS process or inductors mounted externally on the package while only the active devices are integrated on die [44]. Hazucha et al [44] demonstarted a switching VR with 80%-87% efficiency with integrated active devices and on-package inductors ($Q = 20$).

My efficiency analysis assumes a 32nm CMOS process with inductors ($Q= 20$ @ 100MHz) similar to [44]. The design is optimized to achieve a conversion ratio of 1.05V/0.9V at a load current of 16.67A per core (corresponding to total 120W for 8 cores at 0.9V) with an efficiency of 88%. An 8-phase topology is used with 63.5nH inductance per phase. As V_O and load current are reduced, the efficiency of switching VRs decreases monotonically. This is because the switching loss constitutes a higher percentage of the output power as the V_O value reduces. The efficiency is strongly dependent on the operating point at which the switching VR design is optimized. For a design optimized for a higher

V_O, the efficiency at low output voltage drops more rapidly compared to a design optimized for a lower V_O [54]. In Figure 3-5 (a) the on-chip switching VRs are optimally designed for V_O = 0.8V.

3.5 Evaluation

3.5.1 DVFS Algorithms for Efficiency Comparison

In this section, I evaluate the effectiveness of the LDO VRs derived from PCPG devices. To do so, we can use various per-core DVFS algorithms optimized for high-performance multi-core processors including the algorithms exploiting C2C frequency and power variations along with thread migrations (TMs) [5]. For the evaluation, I adopt an integer linear programming (ILP) method for the DVFS algorithms. The ILP formulation is similar to the one used Kim et al [4], which attempts to minimize the power consumption of a multi-core processor for a given performance constraint. I modify the formulation such that it searches for the optimal V_O for each core to maximize $MIPS^3/W$ under a power constraint at each DVFS interval as follows:

Objective:

$$\max imize\left(\sum_{i=1}^{N} MIPS_i = \sum_{i=1}^{N}\sum_{j=1}^{M} IPC_i \times F_{ij} \times x_{ij}\right) \qquad (3\text{-}3)$$

Constraints:

$$\sum_{i=1}^{N}\sum_{j=1}^{M} P_{ij} \times x_{ij} \le P_{totmax} \text{ and } \sum_{i=1}^{N}\sum_{j=1}^{M} x_{ij} \le N \qquad (3\text{-}4)$$

where N is the number of cores; M is the number of V_O steps supported by the DVFS algorithm; $MIPS_i$ and IPC_i are the MIPS and IPC of core i; F_{ij} is the frequency of core i at voltage level j; x_{ij} corresponds to one bit of an M-bit binary variable for core i that is guaranteed to assign core i to only one of M possible V/F states (i.e., $\exists i : \sum_{j=1}^{M} x_{ij} = 1$); P_{ij} is the power consumption of core i, which is a function of $V_O[i]$; P_{totmax} is the allowed total power consumption of the processor; and Eq. (3-3) is the constraint, respectively. In Eq. (3-4), the second constraint is to enforce one V_O selection for each core. The V_I for all LDO VRs is determined by taking the maximum value among $V_O[1]$ to $V_O[N]$.

Teodorescu et al [5]'s algorithm requires manufacturers to store per-core frequency and power values at each voltage level for DVFS algorithms to exploit C2C frequency and power variations. These values can be characterized by the manufacturer and stored, along with many other processor tuning parameters, in a non-volatile memory of the processor. Like other DVFS algorithms, I also need

Table 3-2: Summary of DVFS algorithms explored in this study. "Sh," "Se," "PV," and "TM" indicate "Shared," "Separate," "Process Variation," and "Thread Migration," respectively.

Algorithms	Voltage Domain	Frequency Domain	Process Variation Aware	Thread Migration	Off-Chip VR V_O	On-Chip VR	Constraint
ShV/F	Shared	Shared	No	No	Varying	N/A	$V_{O1} = V_{O2} = \ldots = V_{ON}$
SeV/F	Separate	Separate	No	No	Fixed	SVR	
SeV/F(PV)			Yes	No			
SeV/F(PV/TM)			Yes	Yes			
LDOSeV/F	Virtually Separate	Separate	No	No	Varying	LDO VR	$V_I - V_{Oi} \le 100mV$
LDOSeV/F(PV)			Yes	No			
LDOSeV/F(PV/TM)			Yes	Yes			

to predict workload characteristics like the IPC of each thread to assign a proper V/F to each core for the next DVFS interval. Although I can use various methods to predict the IPC of the next interval based on the current IPC, I assume that the IPC value of each thread at every interval is known in advance (as an oracle method). This is to isolate the impact of the IPC prediction from the $MIPS^3/W$ results so that I can fairly compare the efficacy of the two different VR schemes. Finally, I adopt a simple thread migration (TM) scheme, and assign threads to cores based on IPC and frequency values. For example, the thread with the highest IPC is assigned to the core with the highest frequency at a given voltage (i.e., the fastest core considering C2C frequency variations). Table 3-2 summarizes the DVFS algorithms explored in this study and their constraints. In this study, I assume a baseline processor with a single, chip-wide V/F domain using an off-chip VR (i.e., ShV/F).

3.5.2 Architectural Simulation Environment

I evaluate different DVFS algorithms using a full-system cycle-accurate simulator, GEMS [55]. I modify GEMS to support per-core frequency domains and TM, which requires L1 cache flushing. Table 3-3 summarizes the simulation paramters. The processor configuration contains eight cores. Each core is four wide with 32KB private L1 caches and a shared 512KB L2 cache. The cores are connected to each other using crossbar switches. I evaluate different DVFS algorithms using a full-system cycle-accurate simulator, GEMS [55], after I modify GEMS to support per-core frequency domains and TM requiring L1 cache flushing. In addition to four commercial workloads (Apache, JBB, OLTP, and Zeus, six SPEC OMP V3.2 benchmarks (ammp, applu, art, equake, mgrid, and swim), and four PARSEC benchmarks (Swaptions, X264, Fluid, and Black Scholes) [56], I use five mixes of compute- and memory-bound SPEC2006 benchmarks (eight copies of Bzip2, six copies of Bzip2 and two copies of

Libquantum, four copies of Bzip2 and four copies of Libquantum, two copies of Bzip2 and six copies of Libquantum, and eight copies of Libquantum denoted by 8B0L, 6B2L, 4B4L, 2B6L, and 0B8L, respectively).

3.5.3 Core Frequency and Power Models

Typically, an operating system (OS) determines V/F of cores based on a given power management algorithm, but both the OS and VRs cannot track and respond to instantaneous changes of power consumption. Thus, the OS must conservatively assume the power consumption of cores at each given operating V/F and guaranteed that the entire chip does not exceed its maximum power consumption, if it aims to optimize performance without violating a power constraint at any given moment.

To model the maximum power consumption of cores, I assume that (i) the total maximum power consumption of eight cores is 120W and (ii) 30% of the total power is active leakage [57] at 0.9V. Each core has its own partition of shared L2 cache that shares the V/F domain with the core. Thus, I assume that the L2 power scales with the core power consumption. The power consumption of I/O and other peripheral components including on-chip interconnects, which are tied to other separate fixed V/F domains, is not included in my analysis since it can be regarded as a fixed power cost for all the cases I explore in this study; I/O and on-chip interconnects are responsible for ~15% of the total power in Niagara 2 [58].

Due to WID C2C frequency and leakage power variations, the power consumption of each core differs. To analyze the impact of WID process variations on the frequency and leakage power consumption of each core, I first generate 100 variation maps for threshold voltage (Vth) and effective channel length (Leff) of transistors in a die and characterize frequency and power consumption by following the methodology presented in [42]: WID correlation distance coefficient $\phi = 0.5$ and WID Vth and Leff variations $\sigma_{sys} = 6.4\%$ and 3.2% of the nominal Vth and Leff values, respectively. I apply the Vth and Leff values of each grid point to a FO4 inverter chain and a dummy circuit, which is comprised of 50% inverters, 30% NAND gates, and 20% NOR gates, to obtain the frequency and leakage power scaling factors of each grid, respectively; NAND and NOR gates in a dummy circuit can have up to 4 inputs and their inputs are assigned randomly with either 1 or 0.

Second, I measure the frequency and leakage scaling factors of each grid point at 0.95V to 0.7V using a 32nm technology model and SPICE. I assume that the frequency of each core is determined by the slowest grid point in the core [42] and the frequency of the slowest core is 3.2GHz at 0.9V. Then, each core's maximum dynamic power consumption at 0.9V is $\left(F_i / \sum_{j=1}^{N} F_j\right) \times 120\text{W} \times 0.7$ where F_i and F_j are the frequency of core i and j, and N is the number of cores. With the known frequency, voltage, and dynamic power values, we can calculate the maximum core switching capacitance (i.e., C_{dyn}). This

Table 3-3: Summary of processor simulation parameters.

Fetch/Issue/Retire	4/4/4	# of Cores	8
IL1	32KB/4-way/64B 3 cycles	Branch Predictor/BTB/RAS	YAGS/1K/32
L2	512KB/8-way/64B 10 cycles	DL1	32KB/4-Way/64B 3cycles
Cache Coherency Protocol	Directory-based MESI	Main Memory (size/block/page/latency)	DDR3-1.6GHz 4GB/64B/4KB/7-7-7-20ns
# of MSHRs	8	Write-buffer entries	16

allows us to calculate the dynamic power at any given voltage. The leakage power of each grid point is scaled such that the sum of the leakage power from all grid points in a die is equal to 30% of 120W at 0.9V. The sum of the scaled leakage power from all the grid points belonging to a particular core becomes the core's leakage power.

Finally, I allow some cores to run at V/F higher than 0.9V/3.2GHz as long as the total power constraint is satisfied; this is possible when other cores run at V/F lower than 0.9V/3.2GHz. Since all cores in the baseline processor of this study, which uses a per-chip single VR, run at the same frequency, the dynamic power consumption of the processor is lower than when other processors use per-core V/F domains. Thus, I increase the V/F of the processor until 120W is fully used (i.e., 0.9125V and 3.3GHz).

Note that C2C frequency and power variations change across different dies. However, for this analyses, a typical die map has been picked from the 100 generated maps because a large amount of simulation time is required to repeat the same experiment for hundreds of die maps. Thus, the $MIPS^3/W$ results, which exploit C2C frequency and leakage power variations, represent the value close to the median value of the 100 die maps. Table 3-4 tabulates the frequency and power consumption of

Table 3-4: summary of frequency and power consumption of each core as a function of $V_O[i]$. For each core, the frequency (GHz) and power (Watts) are given in the left and right columns, respectively.

$V_O[i]$	Core 1		Core 2		Core 3		Core 4		Core 5		Core 6		Core 7		Core 8	
0.95V	3.6	15.9	3.8	17.9	4.4	18.2	3.9	16.9	3.8	17.2	4.1	19.3	4.4	29.1	4.1	19.7
0.90V	3.2	12.5	3.4	14.0	4.0	14.5	3.5	13.3	3.4	13.5	3.7	15.1	4.0	22.0	3.7	15.3
0.85V	2.8	9.6	3.0	10.8	3.5	11.3	3.0	10.2	3.0	10.4	3.3	11.7	3.5	16.6	3.3	11.8
0.80V	2.4	7.2	2.5	8.1	3.1	8.6	2.6	7.7	2.4	7.8	2.8	8.8	3.1	12.3	2.8	8.9
0.75V	2.0	5.3	2.1	6.0	2.6	6.4	2.2	5.7	2.1	5.7	2.3	6.5	2.6	9.0	2.4	6.6
0.70V	1.6	3.7	1.7	4.2	2.1	4.6	1.7	4.0	1.7	4.1	1.9	4.6	2.1	6.4	1.9	4.7

each core as function of $V_O[i]$. For each core the frequency (GHz) and power (Watts) are given in the left and right columns, respectively.

3.5.4 MIPS^3/W Comparison

Impact of Limiting $V_I - V_O$ range on MIPS^3/W: Figure 3-6 compares MIPS^3/W of 8-core processors using LDO and switching VRs. The DVFS algorithms with "LDOSeV/F" use LDO VRs while ones with "SeV/F" use switching VRs; see Table 3-2 for the description of the DVFS algorithms. Without exploiting WID C2C process variations and the TM technique, the MIPS^3/W difference between LDOSeV/F and SeV/F is around 2% (3% versus 5% improvement over ShV/F) on average (i.e., geometric mean). However, when WID C2C process variations are exploited, the MIPS^3/W difference between the processors using LDO and switching VRs becomes 1% (14% versus 15% improvement over ShV/F) on average. Finally, the MIPS^3/W difference between the two schemes leads to a 3% difference (22% versus 25% improvement over ShV/F) on average when both the TM technique is applied and WID C2C process variations are incorporated with the DVFS algorithms. Thus, exploiting C2C frequency/power variations and TMs can mitigate the potential limitation of LDO VRs and its relative benefit is higher for processors using LDO VRs.

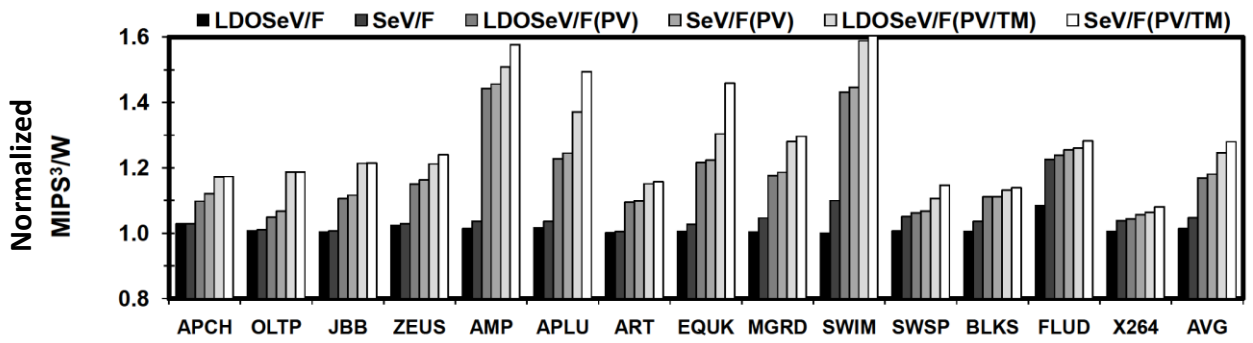


Figure 3-6: MIPS^3/W comparison of 8-core processors supported by LDO (algorithms beginning with the LDOSeV/F prefix) and switching VRs (algorithms beginning with SeV/F). All results are normalized to a processor with ShV/F and do not include the power loss by the off-chip VR. Each interval is comprised of 10-million executed instructions.

Impact of VR efficiency on MIPS^3/W: In Figure 3-6, I did not include the power consumption (i.e., power loss) incurred by both on- and off-chip VRs to isolate to the impact of constraining the VO range for LDO VRs. As explained earlier, the MIPS^3/W difference between processors using LDO and switching VRs is very small. However, since C2C voltage differences are limited to 100mV at each DVFS interval, the differences between the VI value and the VO values must be small. In other words, LDO VRs often exhibit higher efficiency than switching VRs for most DVFS intervals, as shown in Figure 3-5 (b). Consequently, as the efficiencies of both on- and off-chip VRs are considered, a processor using LDO VRs can provide higher MIPS^3/W than a processor using switching VRs as shown in Figure 3-9.

First, the processors using LDOSeV/F and SeV/F, which do not exploit WID process variations, result in worse MIPS^3/W than the processor using ShV/F, which uses only an off-chip VR. This is because the power loss by the on-chip VRs completely negates the benefit of supporting per-core V/F domains for multi-thread applications. Second, when WID process variations are exploited, VSeV/F(PV) and SeV/F(PV) can provide 6% and 4% higher MIPS^3/W than ShV/F on average. I observe that the processor using LDO VRs exhibits higher MIPS^3/W than the one using switching VR. This is the opposite of the trend shown in Figure 3-6, where the power loss by VRs was not considered in computing MIPS^3/W and the MIPS^3/W of the processor using LDO VRs was lower than one using switching VRs. This is mainly due to small C2C voltage variations in multi-thread applications, which allows LDO VRs to provide voltages with higher efficiency than switching VRs as shown in Figure 3-5(b). Finally, when the TM technique is also applied, LDOSeV/F(PV/TM) and SeV/F(PV/TM) can provide 13% and 12% higher MIPS^3/W than ShV/F on average.

Impact of DVFS interval on MIPS^3/W: The interval period for applying DVFS algorithms also impacts the benefit of DVFS. I conservatively assumed the overhead of V/F switching will be the same

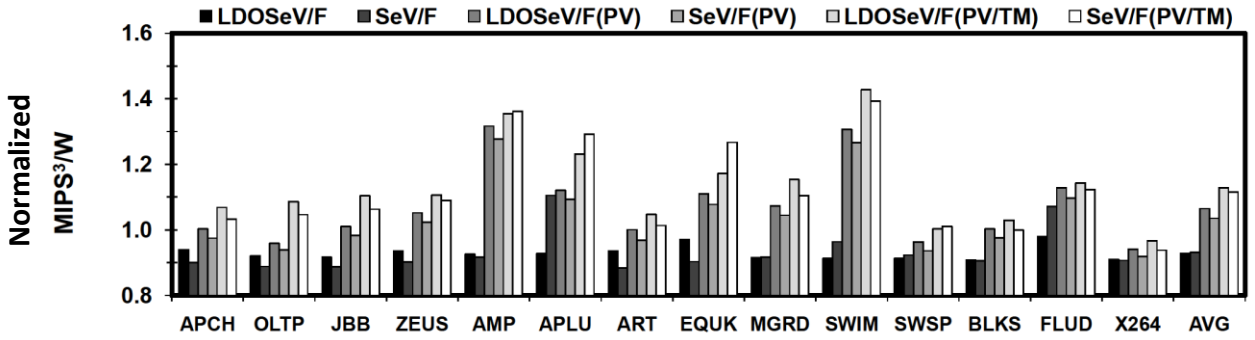


Figure 3-7: MIPS^3/W comparison of 8-core processors supported by LDO (algorithms beginning with the LDOSeV/F prefix) and switching VRs (algorithms beginning with SeV/F) including the power loss by both on- and off-chip VRs. All results are normalized to a processor with ShV/F and include the power loss by the off-chip VR. Each interval is comprised of 10-million executed instructions.

for all DVFS algorithms. Thus, we can factor it out. In theory, shorter DVFS intervals can capture more C2C IPC variations and thus lead to higher performance and power efficiency. Thus, I reduce the DVFS interval to every 5-million instructions while keeping the TM interval at 10-million instructions. As expected, $MIPS^3/W$ for both LDOSeV/F(PV/TM) and SeV/F(PV/TM) increases, but the relative difference between them remains almost the same. Note that the DVFS interval is often determined by considering both (i) the computational overhead of the DVFS algorithm and (ii) the VR efficiency degradation during V_O changing periods [7]. The latter prohibits a very short interval for a simple threshold-based DVFS algorithm even though the current state-of-the-art off-chip switching VRs can support much faster voltages changes. For example, Microsoft Windows Vista uses 20ms for the default value while it could support a more aggressive interval value (e.g., 1ms) for DVFS [59].

Multi-program environment: A processor executing multiple applications can exhibit more substantial C2C IPC variations than when it is running multi-threaded applications, depending on the mix and characteristics of applications. Consequently, supporting a wider range of V_O values using switching VRs may lead to higher $MIPS^3/W$ than using LDO VRs under a specified power constraint. Figure 3-8 shows the $MIPS^3/W$ comparison between two processors using switching and LDO VRs when running five mixes of memory- and compute-bound applications.

First, when the power loss by VRs is not considered, a processor with per-core voltage domains using either switching or LDO VRs has much higher $MIPS^3/W$ than the one using a single chip-wide voltage domain for 6B2L, 4B4L, and 2B6L in Figure 3-8(a). This is due to these applications mixes having much higher C2C IPC variations than the multi-threaded applications. For example, LDOSeV/F(PV/TM) and SeV/F(PV/TM) can provide 34% and 55% higher $MIPS^3/W$ than ShV/F on

average. The processor using LDO VRs provides substantially higher $MIPS^3/W$ than the one using a single chip-wide VR, but 16% lower $MIPS^3/W$ than the one using switching VRs.

Second, when the power loss by the VRs is considered, as shown in Figure 3-8(b), LDOSeV/F(PV/TM) and SeV/F(PV/TM) can yield 24% and 39% higher $MIPS^3/W$ than ShV/F on average. Unlike the multi-threaded applications, LDOSeV/F(PV/TM) results in lower $MIPS^3/W$ than SeV/F(PV/TM), yet the difference between LDOSeV/F(PV/TM) and SeV/F(PV/TM) is reduced to 12%. This is because the power loss by LDO VRs is still lower than switching VRs in many DVFS intervals.

In the previous experiments, I have limited the difference between $V_I - V_O$ to 100mV. This is because the efficiency of LDO VRs becomes lower than switching VRs once the voltage difference becomes larger than 100mV. On the other hand, I found out that forcing such a constraint misses potential power reduction opportunities that can be achieved by operating cores at lower V/F. In other words, the benefit of reducing V/F of cores more can outweigh lower power efficiency of LDO VRs

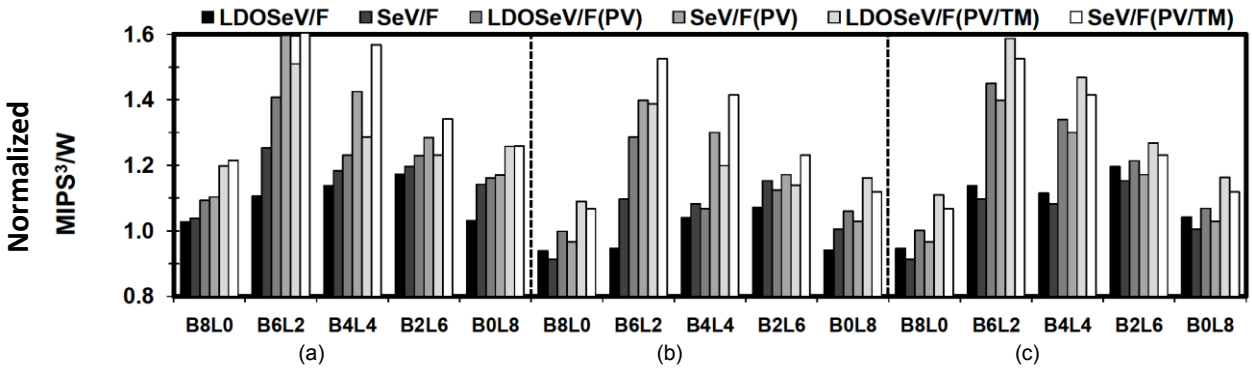


Figure 3-8. $MIPS^3/W$ comparison of 8-core processors supported by LDO (algorithms beginning with the LDOSeV/F prefix) and switching VRs (algorithms beginning with SeV/F). The power loss by VRs is not included in (a) and is included in (b). The $V_I - V_O$ constraint is removed in (c) for LDO VRs. Each interval is comprised of 10-million executed instructions.

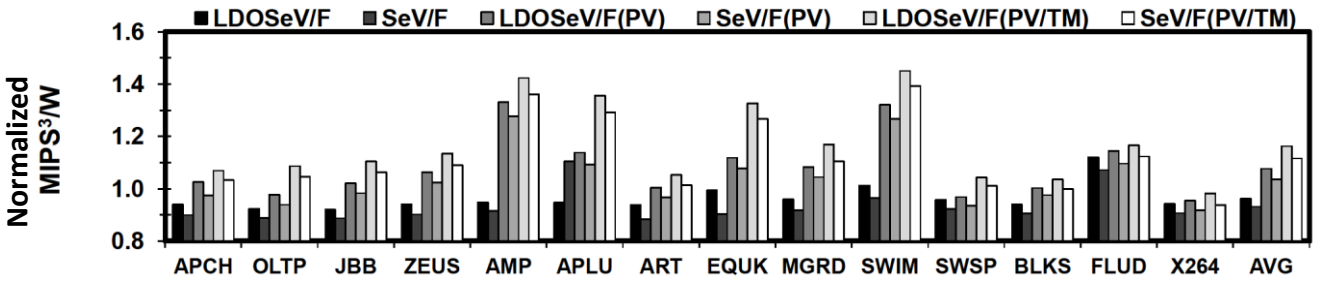


Figure 3-9: MIPS^3/W comparison of 8-core processors supported by LDO (algorithms beginning with the LDOSeV/F prefix) and switching VRs (algorithms beginning with SeV/F) including the power loss from both on- and off-chip VRs. All results are normalized to a processor with ShV/F and include the power loss by the off-chip VR. Each interval is comprised of 10-million executed instructions.

operating at $V_I - V_O$ larger than 100mV. Thus, to evaluate the impact of such a constraint, I remove the $V_I - V_O$ constraint for LDO VRs in Figure 3-8(c). When $V_I - V_O$ is larger than 100mV, the power loss by LDO VRs is higher than that of switching VR. However, the power loss by LDO VRs becomes lower than that of switching VRs for the DVFS intervals exhibiting $V_I - V_O$ less than 100mV. Consequently, as long as I have more DVFS intervals with $V_I - V_O$ less than 100mV, the processor using LDO VRs can lead to higher MIPS^3/W than the one using switching VRs. Figure 3-8(c) shows that LDOSeV/F(PV/TM) has 4% higher MIPS^3/W than SeV/F(PV/TM). To validate this result, I analyzed the fraction of DVFS intervals exhibiting $V_I - V_O$ more than 100mV. For B4L4, I measured the fraction of DVFS intervals in which LDO VRs have lower efficiency than switching VRs after the V/F and core power consumption profiles obtained from SeV/F(PV/TM) are applied to both efficiency functions of LDO and switching VRs. This reveals that LDO VRs show higher efficiency than switching VRs for close to 60% of the total DVFS intervals that are experienced by individual cores.

Impact of Removing $V_I - V_O$ range constraint on MIPS^3/W of Multi-threaded Workloads: Figure 3-9 compares MIPS^3/W of 8-core processors using LDO and switching VRs after the voltage range constraint is removed. Although WID C2C process variations are not exploited and the TM

technique is not applied, LDOSeV/F provides 4% higher $MIPS^3/W$ than SeV/F on average. When both WID C2C process variations and TM are exploited, LDOSeV/F and SeV/F provide 16% and 12% higher $MIPS^3/W$ than ShV/F on average; LDOSeV/F leads to 4% higher $MIPS^3/W$ than SeV/F whether or not WID C2C process variations and/or TM are exploited. This is because LDO VRs exhibit higher efficiency than switching VRs for most DVFS intervals. This is mainly due to small C2C voltage variations in multi-threaded applications, which allows LDO VRs to provide voltages with higher efficiency than switching VRs, as shown in Figure 3-5.

3.6 Chapter Summary

In this chapter, I propose a cost-effective technique to support per-core voltage domains for high-performance server-class processors. I demonstrate that PCPG devices augmented with small circuitry can operate as low-cost LDO VRs. Unlike on-chip switching VRs, LDO VRs do not require on-chip inductors, which are expensive and a major technical challenge for practical use of on-chip switching VRs, but their efficiency becomes poor as their output voltage applied to cores drops (i.e., large difference between input and output voltage of the LDO VRs). Consequently, per-core DVFS using LDO VRs may lead to lower performance and power efficiency than using switching VRs. However, my experiments show that C2C voltages variations are relatively small when the voltages are optimized to maximize performance under a power constraint. By modeling the power efficiency of both LDO and switching VRs at 32nm technology, I show that the $MIPS^3/W$ of an 8-core processor using LDO VRs is slightly higher than that of a processor using switching VRs. This is due to the higher efficiency of LDO VRs compared to switching VRs for small C2C voltage variations.

Chapter 4

Low-Voltage On-Chip Cache Architecture using Heterogeneous Cell Sizes

Although dynamic voltage and frequency scaling (DVFS) has been a very effective power management technique to minimize the power consumption of multi-core processors, its efficiency has been continuously decreased due to the shrinking voltage scaling window. The ever-increasing process variability in nanoscale technology such as random dopant fluctuation (RDF) and line edge roughness (LER) limits the voltage scaling of on-chip memory elements to some minimum operating voltage, VDDMIN [60]. At lower voltages, multi-core processors cannot operate reliably due to defective memory cells that fail to maintain their correct states. As a result, the failure probability of individual memory cells in large-scale structures such as last-level caches (LLCs) often determines the VDDMIN for the processor.

The simplest approach to lower the VDDMIN for a target yield is to increase the size of the transistors in the static random access memory (SRAM) cells. This makes the transistors less sensitive to mismatches induced by process variations, which is the cause of the higher cell failure rates at low voltages. However, in current state-of-the-art multi-core processors, LLCs already occupy more than 50% of the total die area. Furthermore, the capacity of such LLCs needs to be increased as more cores are integrated into a single die. This makes it impractical to substantially increase the size of individual SRAM cells.

In this chapter, I develop an LLC architecture that exploits a more detailed understanding of the operating characteristics of processors using DVFS. To develop such an understanding, I run experiments using a commercial computing system with an AMD® high-performance 4-core processor and a benchmark suite that mimics the typical usage of server applications (i.e., SPECPower Sever Side Java (SSJ) 2008 [61]). The experimental results reveal that the processor spends a substantial fraction of its runtime in high frequency/voltage states when the load level is higher than 50%. For example, if the load level is higher than 50%, then more than 40% of the run-time is spent in the highest voltage/frequency state.

I exploit these characteristics to deliver both high-performance and low V_{DDMIN} by architecting an LLC consisting of a spectrum of cell sizes. For low-power operation, I exclusively use large cells that exhibit low failure rates at low voltages; therefore, the LLC can operate at a low V_{DDMIN}. On the other hand, for high-performance operation, I operate at a high enough voltage that the failure rate of even the small cells in the LLC is sufficiently low for their use, providing the needed LLC capacity. As operating voltage decreases, subsets (e.g., ways in a set-associative LLC) of cells are disabled in order of size, starting with the smallest. Because a processor must run at lower frequencies at lower voltages, the frequency gap between on-chip cores and off-chip memory decreases. As a result, the performance penalty of having a smaller effective LLC capacity in low voltage/frequency states is much smaller than it would be in high voltage/frequency states. Note that a conventional architecture needs to use large cells across the entire LLC to provide the same low V_{DDMIN} as my proposed architecture. This will either require a larger die area for the LLC capacity or result in a smaller LLC capacity for the same die area.

Unlike the prior techniques that disable only defective cache lines [21] [22], my proposed technique does not require expensive on-chip fuses [62] to record the locations of the defective cache lines identified during a manufacturing test. Although on-line LLC testing mechanisms based on error-correction coding (ECC) can be adopted to avoid the use of fuses [63], their capability is limited by the strength of the ECC. Furthermore, the prior techniques can result in significant performance variations across processor chips. Since defective lines may have different performance impact due to their locations [64]. In contrast, my technique provides deterministic performance behavior because each voltage/frequency operating state leads to a deterministic LLC size for all the processors from different dies. Finally, my technique can be transparently combined with other existing techniques using repairing schemes [22] [62] and ECCs [65] that are used to support a low V_{DDMIN}.

The specific contributions of this study are as follows. First, I architect cost-effective LLCs consisting of heterogeneous cell sizes to provide both high-performance and a low V_{DDMIN}. Second, I analyze the impact of the LLC architectures on the area, performance, and power consumption of a processor that adjusts voltage and frequency periodically due to a changing workload demand. Third, I provide a micro-architectural technique that reduces the performance impact of writing back dirty lines when disabling subsets of the LLC due to voltage/frequency changes.

The remainder of this chapter is organized as follows. Section 4.1 analyzes the impact of LLC capacity on performance at different processor frequencies. Section 4.2 demonstrates the trade-off between cell size, voltage, and failure probability. Section 4.3 describes my proposed LLC architectures and the micro-architectural techniques used to maximize their performance. Section 4.4 details the experimental methodology and results. Section 4.5 summarizes this chapter.

4.1 Impact of LLC Size on Performance versus Processor Frequency

One of the major reasons for using a large LLC is to minimize the number of accesses to off-chip memory. Retrieving off-chip data incurs large latency penalties due to the large frequency gap between fast on-chip cores and slow off-chip memory. Therefore, a large on-chip cache capacity is essential to achieving high-performance. However, at lower voltage and frequency of the processor are decreased, such as through DVFS, the frequency gap between on-chip cores and off-chip memory decreases. This reduces the importance of having a large LLC.

Figure 4-1 plots the data of relative runtime versus core frequency for different L3 cache capacities. I used 4 commercial workloads [66] running on the GEMS multi-core simulator [55]: (a) Apache, (b) JBB, (c) OLTP, and (d) Zeus; see Table 4-2 in Section 4.4.1 for the simulation parameters. I first warmed up the caches with 100 million instructions and ran one billion instructions. Next, I measured the total execution time of each workload with the default LLC cache capacity at each frequency state, i.e., 2.4 GHz, 1.6 GHz, 1.2 GHz, and 0.8 GHz. Finally, I decreased the L3 cache capacity gradually as I decreased the supply voltage (and thus operating frequency) to obtain the total execution time for the different LLC capacities. While the runtime degradation is notable (3% - 6%) for small on-chip cache capacity at high frequency (e.g., 25% of 8 MB at 2.4 GHz), it becomes negligible (~1% or less) at low frequency (0.8 GHz-1.2 GHz). This implies that I can use a smaller LLC capacity at lower voltage and frequency operating states without causing a noticeable increase in runtime.

The runtime degradation shown in Figure 4-1 is the average for executing 1 billion instructions. The “average” runtime degradation (~6%) for smaller L3 capacities might appear to be a small perfor-

mance impact. However, average runtime degradation in high-performance mode (i.e., highest voltage/frequency) can be misleading. For example, 21% (Apache), 23% (OLTP), and 13% (Zeus) of total execution intervals shows 10% (or more) runtime degradation when the L3 capacity was reduced to 4 MB; I measured runtime for every 10 million instructions for the different L3 sizes. This relatively high-performance degradation in certain execution intervals may not be acceptable for many quality-of-service critical applications, emphasizing the need for a large LLC capacity in high-performance mode.

My LLC architecture, which consists of heterogeneous cell sizes, is specifically designed to take advantage of this effect. I use only the largest cells at voltages close to the VDDMIN (effectively operating an LLC of smaller capacity) and all cells at the highest, high-performance, voltage settings. At the lower voltages, the decreased frequency gap between on-chip cores and off-chip memory means that the negative performance impact of the decreased effective capacity of the LLC is negligible, while the use of smaller cells at higher voltages yields significant savings in die area. In contrast, conventional LLCs are designed to operate the whole LLC reliably at all allowable operating voltages/frequencies.

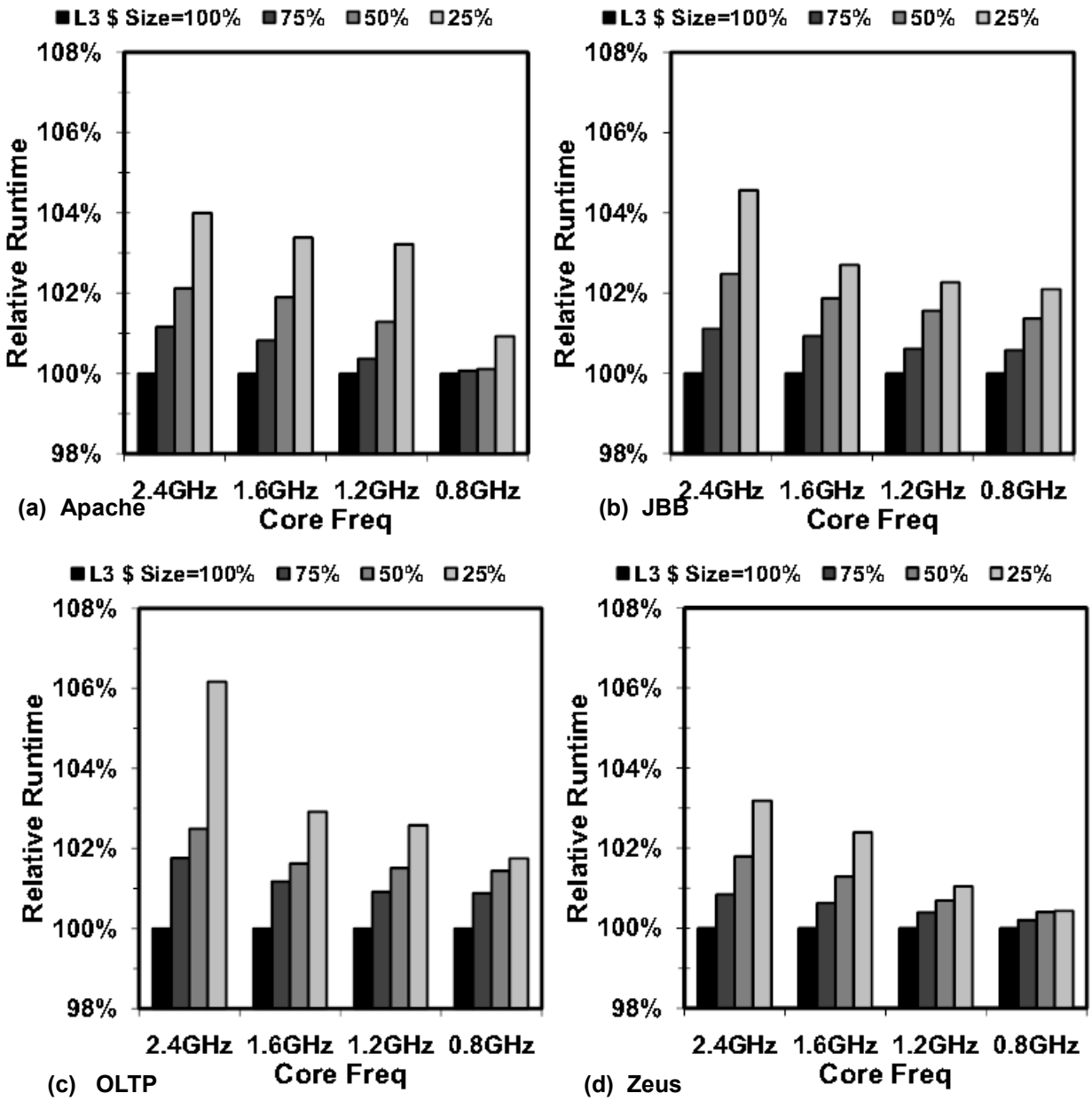


Figure 4-1: Relative runtime versus core frequency for different L3 cache capacities (100%, 75%, 50%, and 25% of an 8 MB L3 cache). The runtime values are normalized to the runtime with 100% L3 cache capacity at each core frequency after executing 1 billion instructions.

4.2 SRAM Cell Failure Probability and V_{DDMIN} versus Cell Size

4.2.1 Impact of Transistor's Size on RDF and LER

The main source of SRAM cell failures at low voltages is due to RDF and LER [60]. These result in device mismatches in SRAM cells that employ symmetric, crosscoupled circuit topology, making such cells unstable at low voltage. The degree of device mismatches, i.e., the standard deviations of each transistor's threshold voltage (V_{TH}) and channel length (L) are inversely proportional to device size as described in [67]: " $\sigma V_{TH} \propto 1/\sqrt{W \times L}$ " and " $\sigma L \propto 1/\sqrt{a + W}$ " where a is a technology-dependent constant, W is transistor channel width, and L is transistor channel length, respectively. As a result, typical SRAM cells that use minimum geometry transistors to improve integration density also exhibit a large amount of device mismatch, which increases the failure probability of SRAMs at low voltage. Thus, to achieve a low V_{DDMIN} for a target yield, on-chip caches must use larger transistors that exhibit less variation and/or special circuit and logic techniques, e.g., read/write assist, redundancy, ECC, etc. [68] [13] [14] that mitigate the negative impact of process variation on SRAM failures.

4.2.2 Impact of Cell Size on Cell and Cache Failure Probabilities

Zhou et al [69] design and optimize six different 6-transistor SRAM cells (i.e., C1~C6 cells). They analyzed the failure probabilities of the cells due to process variations at 32nm. These analyses demonstrated that increasing a cell's size can reduce its failure probability by orders of magnitude. For this study, I take the failure probabilities of their presented cells [69]. I then scale them to provide V_{DDMIN} = 0.7V for an 8MB LLC using C6 cells. In Table 4-1, I show the optimized area for each cell

(relative to the area of the C1 cell), as well as cell size versus the failure the probability of a given cell ($P_{FAILCELL}$) at different operating voltages.

Note that $P_{FAILCELL}$ decreases by orders of magnitude at the same voltage as cell size increases; the $P_{FAILCELL}$ for the largest two cells (C5 and C6) are very close to the Intel®'s data at 45nm technology [22]. The cache is said to fail when it contains at least one bad cell. Given a $P_{FAILCELL}$ value for a cell size and one voltage, the failure probability of the cache, $P_{FAILCACHE}$ can be calculated as:

$$P_{FAILCACHE} = 1 - (1 - P_{FAILCELL})^n \qquad (4\text{-}1)$$

where *n* is the number of cells in the cache. As the number of cells (i.e., cache size) increases, the overall $P_{FAILCELL}$ increases as well. This limits the scaling of SRAM cell size since larger LLCs are increasingly required for high-performance multi-core processors.

4.3 LLC Architecture using Heterogeneous Cell Sizes

An LLC consists of hundreds or thousands of SRAM sub-arrays. My approach uses multiple cell

Table 4-1: Cell size and VDD versus $P_{FAILCELL}$ at different operating voltages.

	C1	C2	C3	C4	C5	C6
Relative Cell Size	1.00	1.12	1.23	1.35	1.46	1.58
$P_{FAILCELL}$ @ 0.90V	3.207×10^{-7}	2.532×10^{-9}	6.980×10^{-11}	4.517×10^{-12}	5.054×10^{-13}	1.217×10^{-13}
$P_{FAILCELL}$ @ 0.85V	5.413×10^{-7}	1.033×10^{-8}	3.113×10^{-10}	1.550×10^{-11}	3.846×10^{-12}	1.021×10^{-12}
$P_{FAILCELL}$ @ 0.80V	1.015×10^{-6}	3.028×10^{-8}	1.513×10^{-9}	7.602×10^{-11}	2.902×10^{-11}	8.952×10^{-12}
$P_{FAILCELL}$ @ 0.75V	2.017×10^{-6}	8.136×10^{-8}	7.429×10^{-9}	4.095×10^{-10}	2.176×10^{-10}	7.888×10^{-11}
$P_{FAILCELL}$ @ 0.70V	4.144×10^{-6}	2.121×10^{-7}	3.724×10^{-8}	2.247×10^{-9}	1.630×10^{-9}	6.995×10^{-10}

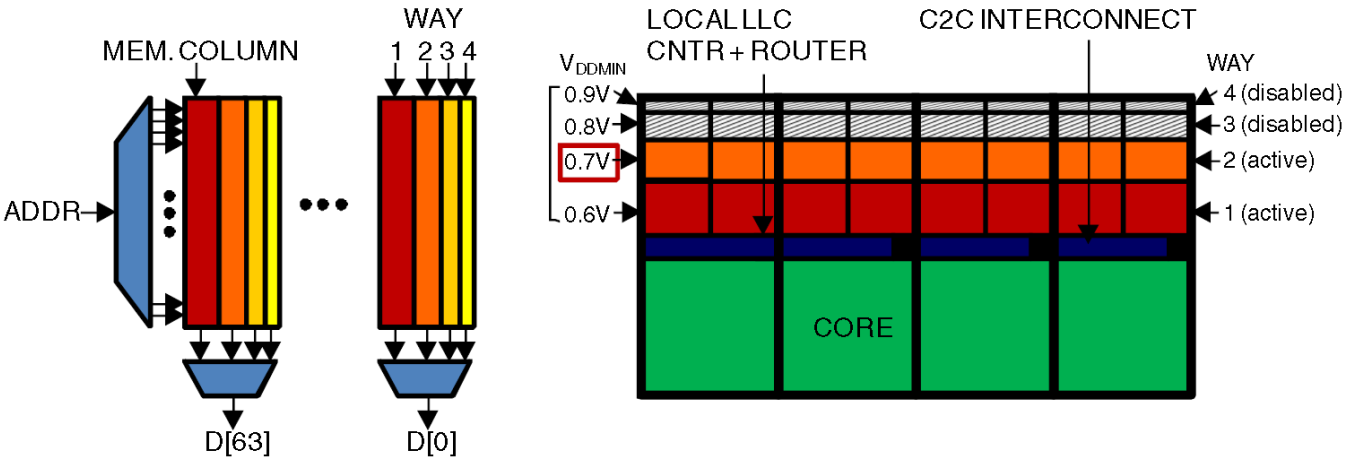
sizes in a single LLC. When high-performance is demanded, the processor runs at high voltage/frequency states where even small cells can operate reliably. As supply voltage is lowered, the failure rate of small cells increases exponentially (cf. Table 4-1) and I disable ways or sets one after another beginning with those consisting of the smallest SRAM cells. Since, as discussed in Section 4.1, at lower voltage/frequency states the frequency gap between on-chip cores and off-chip memory is also smaller, not using the smallest cells has little impact on performance. Meanwhile, ways or sets that are implemented with large cells remain active (and reliable) at lower supply voltage, providing the needed LLC capacity. In this section, I first present my LLC architectures with heterogeneous cell sizes along with the analysis of the probability of LLC failure as a function of the specific LLC design. Second, I discuss the micro-architectural implications and performance impacts of disabling a part of the LLC, and propose various micro-architectural mitigation techniques.

4.3.1 LLC Implementation using Heterogeneous Cell Sizes to Support Low V_{DDMIN}

Figure 4-2 (a) shows an example of implementing a subarray of a four-way set-associative LLC with heterogeneous cell sizes. Each column maintains the same height but becomes wider to accommodate larger cells since the cell size increases in the horizontal direction [69]. Each group of four columns shares one sense-amplifier (i.e., one output bit). Each column in the group represents a way in the four-way set-associative LLC. Note that this type of set-associative LLC implementation has been used in commercial processors since it can protect a cache set against a particle strike flipping multiple neighboring bits [70]. Figure 4-2 (b) illustrates another example of building a four-way set associative LLC, where each group of subarrays is associated with each way (and each cell size). In this illustration, the total number of sub-arrays will be divided into four groups where each group

represents a particular way with a particular cell size. In Figure 4-2 (b), the sub-arrays with larger cells become taller since I assume that the sub-arrays are rotated by 90 degrees for the sake of presentation. Also, the processor and LLC are operating at 0.7V. Thus, the LLC sections corresponding to ways three and four are disabled at 0.7V since they are comprised of small cells, many of which will fail at such a voltage.

In order to estimate a required level of V_{DDMIN}, I take an AMD® Opteron quad-core processor (Model: 2378) supporting 2.4GHz, 1.6GHz, 1.2GHz, and 0.8GHz voltage/frequency states for DVFS. According to the circuit-level analysis using a 32nm predictive technologies model (PTM) and FO4 (i.e., Fan Out of 4) inverter chains, the voltage values for 1.6GHz, 1.2GHz, and 0.8GHz are respectively close to 0.8V, 0.75V, and 0.7V, if 0.9V (nominal voltage of the 32nm PTM [71]) can



(a) Sub-array with 4 different cell sizes (b) LLC with way 3 and 4 disabled at 0.7V

Figure 4-2: (a) Sub-array with 4 different cell sizes. (b) LLC architecture using 4 different cell sizes to support an area-efficient low V_{DDMIN} LLC where way 3 and 4 are disabled at 0.7V

provide 2.4GHz. Thus, for this study, a V_{DDMIN} of 0.7V is assumed. I build a baseline 8MB LLC consisting of only C6 cells (i.e., a homogeneous LLC consisting of only the largest cells).

The heterogeneous LLC designs I consider consist of m different types (sizes) of cells with n(i) cells of each type, i=1, 2, ..., m. To compute $P_{FAILCACHE}$ at a particular voltage, Eq. (4-1) becomes:

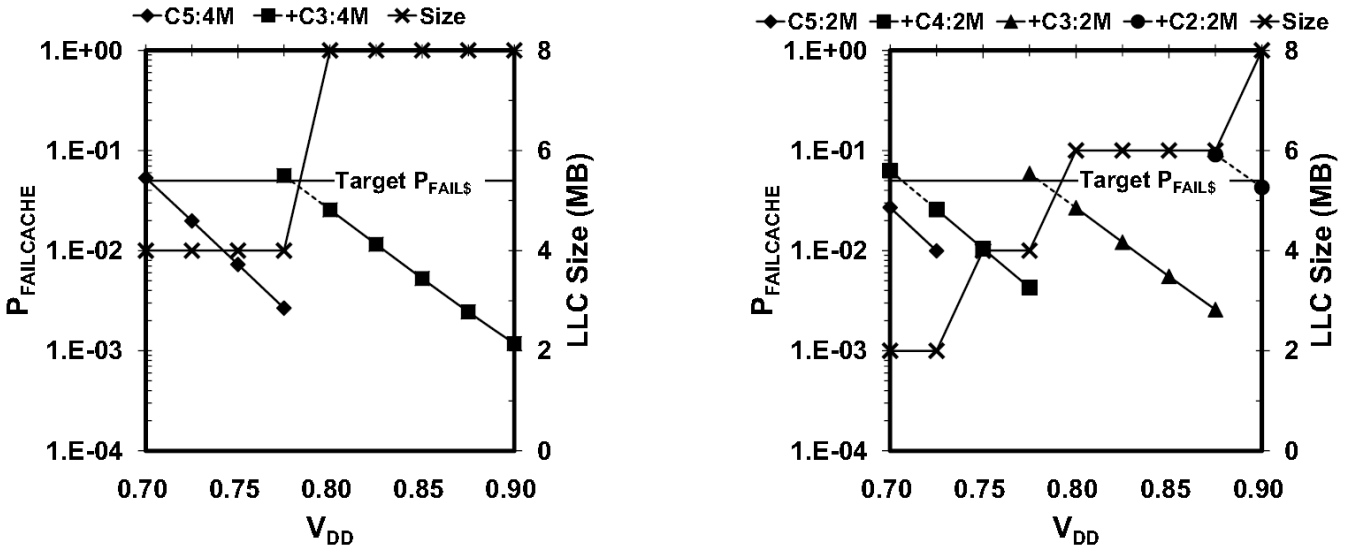
$$P_{FAILCACHE} = 1 - \prod_{i=1}^{m} (1 - P_{FAILCELL}(i))^{n} \qquad (4\text{-}2)$$

where $P_{FAILCELL}(i)$ is the $P_{FAILCELL}$ of cell size i at a given voltage. In Figure 4-3, I show the $P_{FAILCELL}$ and effective LLC size (or capacity) versus voltage for two different LLC architectures with heterogeneous cell sizes.

Figure 4-3 (a) shows $P_{FAILCACHE}$ of a LLC comprised of C5 and C3 cells (i.e., m=2) where each cell size provides 4MB capacity (i.e., n(1) = n(2) = 4×8×220). In this particular architecture, I can reduce the total cell area by 15%, i.e., total LLC area by 13% considering SRAM array efficiency equal to 85% [72]. When the voltage (frequency) is higher than 0.8V (1.6GHz), the processor will be able to use the full 8MB LLC capacity, satisfying the yield target that requires $P_{FAILCELL}$ to be below 0.05. If the voltage (frequency) is below 0.8V (1.6GHz), the 4MB section of the LLC consisting of the smaller C3 cells will be disabled.

As a consequence, the overall $P_{FAILCACHE}$ will not satisfy the yield target as shown in the dotted-line segment of the "+C3:4M" curve in Figure 4-3 (a). However, the 4MB section consisting of C5 cells will operate reliably, meeting the yield target at the voltage range, i.e., [0.7V, 0.9V) as presented in the "C5:4M" curve. Effectively, m=1 in (2) when voltage is below 0.8V and m=2 in Eq. ((4-2) when voltage is above 0.8V. However, I can satisfy the similar $P_{FAILCACHE}$ using the smaller cell size, i.e., C5 since the total number of C5 cells are half of the baseline with C6 cells. To see this formally, I simplify Eq. (4-2) as follows:

$$\begin{aligned} P_{FAILCELL} = \; & 1 - \prod_{i=1}^{m}\left(1 - P_{FAILCELL}(i)\right)^{n(i)} = \\ & 1 - \prod_{i=1}^{m}\exp\left\{n(i) \times \log\left(1 - P_{FAILCELL}(i)\right)\right\} = \\ & 1 - \exp\{-\sum_{i=1}^{m} n(i) \times (P_{FAILCELL}(i))\} \end{aligned} \tag{4-3}$$



(a) C5/C4 (4MB/4MB) LLC

(b) C5/C4/C3/C2 (2MB/2MB/2MB/2MB) LLC

Figure 4-3: $P_{FAILCACHE}$ and LLC size versus voltage. (a) 4MB with C5 for 0.7~0.9V and 8MB with C5+C3 for 0.8~0.9V. (b) 2MB with C5 for 0.7~0.9V, 4MB with C5+C4 for 0.75~0.9V, 6MB with C5+C4+C3 for 0.8~0.9V, and 8MB with C5+C4+C3+C2 for 0.9V.

where the approximation is the first term of a Taylor series, and is quite accurate for the small failure probabilities of the cells considered herein. This approximation tells us that the overall LLC failure probability is a function of a weighted average of the individual cell probabilities, weighted by the number of cells of each size. Thus, an LLC consisting of 8×8×220 C6 cells, i.e., 8MB would have approximately the same cache failure rate as an LLC consisting of 4×8×220 C5 cells if the failure probability of an individual C5 cell were twice that of a C6 cell. Note that the baseline 8MB LLC is designed with C6 to achieve the required $P_{FAILCELL}$ for a yield target at V_{DDMIN} = 0.7V, although using the smallest cell (i.e., C1) should be sufficient to satisfy the access time constraint. In other words, meeting the $P_{FAILCELL}$ constraint at the target V_{DDMIN} forces us to use a large cell (i.e., C6).

To minimize the LLC area further, I can design a more heterogeneous LLC. Figure 4-3 (b) provides $P_{FAILCELL}$ of a LLC composed of C5, C4, C3, and C2 cells where each cell size gives 2MB of capacity (for compactness of notation I refer to this as a 2MB/2MB/2MB/2MB C5/C4/C3/C2 LLC). As the voltage is decreased from 0.9V to 0.8V, to 0.75V, and to 0.7V, the LLC capacity is reduced from 8MB to 6MB, to 4MB, and to 2MB. In this architecture, $P_{FAILCELL}$ is close to 0.05 for the full 8MB capacity at 0.9V. As the voltage decreases to 0.8V, 0.75V, and 0.7V, each 2MB section consisting of C2, C3, and C4 cells will, respectively, be disabled in turn. Within their range of valid operating voltages the resulting PFAILCACHE of each of the 6MB, 4MB, and 2MB sections of the LLC is less than 0.05, satisfying the yield target. Using this architecture, I can reduce the total area dedicated to SRAM cells by 18%, i.e., total LLC area by 16% if I assume 85% array efficiency. In this study, I also provide two more LLC architectures: 1) a 4MB/2MB/2MB LLC consists of C2/C3/C4 cells, and 2) a 2MB/2MB/4MB consists of C1, C2, and C4 cells. These two additional LLC architectures satisfy the yield target for the given voltage range, 0.7V-0.9V as long as the proper section of the LLC is

LLC Arch.	Capacity, V_{DDMIN}, and relative area associated w/ each cell type in LLC	Rel. tot. area
Baseline	C6:8M:0.7V	1.00
A	C5:2M:0.7V | C4:2M:0.75V | C3:2M:0.8V | C2:2M:0.9V	0.81
B	C5:4M:0.7V | C3:4:0.8V	0.85
C	C5:2M:0.7V | C4:2M:0.75V | C3:4:0.8V	0.83
D	C5:4M:0.7V | C3:2M:0.8V | C2:2M:0.9V	0.83

Figure 4-4: Total LLC cell area for different LLC configurations relative to the baseline. In each colored box whose area is proportional to the total cell area for given cell size, X:Y:Z represents cell size, capacity, and minimum operating voltage.

shutdown for each voltage downtransitions. See Figure 4-4 for the total LLC cell area and the operating voltage rage of each section for four different LLC architectures relative to the baseline 8MB one.

Finally, note that my proposed LLC architectures are based on 6-transistor SRAM cells. However, they can be also applied to LLCs implemented with other types of SRAM cells (i.e., 8- or 10-transistor SRAM cells). This is because these SRAM cells exhibit the same fundamental trade-off between cell size and failure probability; 10-transistor SRAM cells show much lower failure probabilities but they are larger than either 6- or 8-transistor SRAM cells.

4.3.2 Micro-Architectural Techniques for LLC Way Shutdown

In Figure 4-2, as supply voltage decreases, one LLC way after another will be disabled in ascending order of cell size. Since all cells in a given column share a vertical VDD line in modern SRAM cell designs, they can easily be turned off with an existing SRAM sleep mechanism [73] in Figure 4-2 (a). When a voltage/frequency down-transition is triggered by DVFS, an LLC way that cannot operate reliably at the new voltage is shut down. In such a case, the dirty LLC lines in that LLC way must be written back to the main memory.

Note that the mechanism for shutting down a subset of LLC is already available in commercial multi-core processors to reduce leakage power consumption [74]. Figure 4-5 illustrates the shutdown process for an LLC way and how an LLC access request is handled during the process. Once the DVFS controller decides to decrease the operating voltage/frequency of the processor, each local LLC controller examines each line in the way that is being shut down. If the line is dirty, it is either (a) written back to the memory controller queue (i.e., cache to memory (C2M)) or (b) moved to another way after evicting a least recently used clean line in the same set (i.e., cache to cache (C2C)). The next line is then examined after the status bit of the dirty line is set to the "invalid" state. This process is repeated until all lines are examined in the way that needs to be shut down. Note that a way shutdown process using option (a) may increase the traffic between on-chip cores and off-chip memory (and thus power consumption).

The LLC can still service read/write requests to minimize the performance impact associated with the shutdown operations. Say that a read or write operation is requested to the LLC. First, the flush operation for the next line is interrupted after finishing the current one (cf. (1) in Figure 4-5). Second, the set corresponding to the read or write address is accessed (cf. "(2) RD/WR REQ" in Figure 4-5) where the line was already examined. Third, for "(3) RD/WR OP" in Figure 4-5, if the line was clean, the read request returns the data to the lower-level cache, i.e., an LLC hit. However, if the line was dirty (thus invalidated in the current LLC way), (a) always incurs a LLC miss but (b) may result in a LLC hit since the previous dirty line could have been moved to another way. For a write operation, if the line was clean, an LLC miss is forced since the line has been already examined and the modified data line cannot be written back anymore. If it was dirty, and similar to the read access case, it can incur an LLC miss or hit depending on a flushing scheme. Finally, the LLC controller resumes the

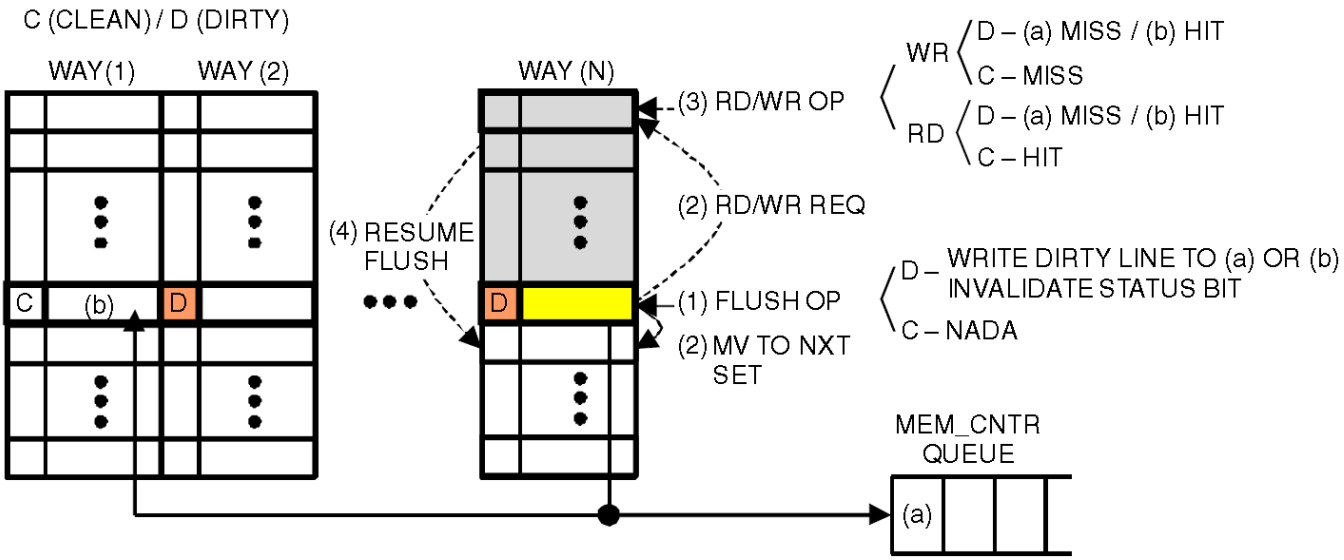


Figure 4-5: LLC way shutdown process and LLC access request handling during the shutdown process: (a) writes back the dirty line to the memory queue and (b) moves the dirty line to another way after evicting a least recently used clean line in **the same set.**

flush operation after completing the service for the request (cf. "(4) RESUME FLUSH" in Figure 4-5). Note that a read or a write operation to a line that has not been examined can be serviced like accessing a normal LLC line.

4.3.3 Performance and Power Impacts of Heterogeneous LLC Architectures

The heterogeneous LLC architectures may affect performance and power both positively and negatively. First, the leakage power remains significant due to the large number of cells although the LLC is implemented with low-leakage devices and its temperature is lower than the on-chip core area. The heterogeneous LLC architectures can reduce a substantial amount of the LLC leakage power since 1) some LLC ways are automatically disabled at low voltage/frequency operating states and 2) they are implemented with smaller cells that consume less leakage power than larger ones. The heterogeneous LLC architectures also require significantly less die area for the same capacity (Figure 4-4). This freed-

up die area can, in turn, be used to increase the LLC capacity, providing higher peak performance at the highest voltage/frequency state.

On the downside, both the flushing operations required before reducing voltage/frequency and disabling LLC ways and the reduced LLC capacity at low voltage, increase the number of accesses to the LLC and off-chip memory. These effects reduce overall performance and increases memory system power consumption. However, one should note first that workloads running on high-performance multi-core processors spend a substantial fraction of their run-time at high voltage/frequency states, and second that the interval of voltage/frequency changes is often longer than 10ms in a commercial operating system (OS) such as Microsoft Windows [75] [76]. In addition, both the flushing operations and the LLC capacity reductions occur only when voltage/frequency decreases (and not when voltage/frequency increases). Therefore, both the performance and power impacts should be quite small. A detailed analysis of the impact of the flushing operations and reduced LLC capacity for each request to change the voltage/frequency level per interval is provided in Section 4.4.2.

Comparing the two heterogeneous LLC architectures A and B of Figure 4-4, I observe a trade-off between area and performance/power impact. Architecture B requires more area than A, but B shuts down the LLC ways only when there is a voltage/frequency down-transition from 0.8V to 0.75V. On the other hand, architecture A requires the shutdowns of the LLC ways whenever there is any voltage/frequency down transition. Thus, architecture B will result in fewer accesses to the LLC and off-chip memory (with less LLC leakage reduction) than will A. Two other heterogeneous LLC architectures (C and D in Figure 4-4) are positioned between A and B in terms of the area and performance/power impact. Either can be preferred depending on which voltage/frequency state the processor spends more time in. Finally, since SRAM cell size affects the access time of SRAM arrays, it may

be argued that using small cells increases LLC latency (and thus decrease performance). However, unlike L1 caches, providing a larger capacity with smaller die area has been a more critical design priority than a shorter latency for architecting LLCs. Furthermore, the overall impact of using small cells on latency is very small because the long latency of LLCs (e.g., more than 30 processor cycles in Intel® i7) is mainly due to address and data distributions across the structures.

Thus, if there is no V_{DDMIN} constraint, C1 cells would be most probably used for implementing conventional LLCs. Finally, recall from Section 4.1 that the height of cells is fixed and only the width is varied depending on the size of the transistors. Thus, the size of arrays with larger cells grows in the word-line direction, resulting in reduced word-line delay. On the other hand, smaller cells often exhibit more bit-line delay. Overall, since the increased bit-line delay of smaller SRAM arrays is countered by the reduced word-line delay, the overall SRAM array access time remains nearly the same. According to my simulation, an array with C3 cells showed less than a 5% increase in the total delay of the word-line and bitline, relative to one consisting of C6 cells; other components of an SRAM array such as pre-charger, row/column decoder, sense amplifier also significantly contribute to the access time of the SRAM array. Hence, the impact on overall latency of small increases in the total delay of the word-line and bit-line by using small cells should be negligible.

Table 4-2: Simulation parameters.

Core Frequency/Voltage	2.4~0.8GHz/0.9~0.7V	**# of Cores**	4
Fetch/Issue/Retire	4/4/4/	**Branch Predictor/BTB/RAS**	YAGS/1K/32
Private IL1	32KB/4-way/32B (2 cycles)	**Private DL1**	32KB/4-Way/32B (2 cycles)
Private L2	256KB/8-way/64B (10 cycles)	**Shared L3/core**	8MB/16-way/64B (36 cycles)
Cache Coherency Protocol	Directory-based MESI	**Main Memory (size/block/page/latency)**	DDR3-1.6GHz 4GB/64B/4KB/7-7-7-20ns
MSHR	8	**Write-buffer**	16

4.4 Evaluation

4.4.1 Simulation Environment

The configuration of my processor is similar to that of an Intel® i7 multi-core processor containing four cores with a shared 8M L3 cache. Each core is four wide with a private L1/L2 cache hierarchy. I evaluate my proposed LLC architecture using full-system cycle-accurate simulation using and GEMS [55] after I augment a shared L3 cache in GEMS. I simulate four commercial workloads [66] and two memory bounded SPEC2006 programs (i.e., sphinx and libquantum) [77] with the simulation parameters presented in Table 4-2. In implementing a DVFS mechanism for the architectural simulation, I assume that all cores share a single voltage domain and use a threshold-based algorithm similar to the one appeared in [78] for 1ms, 5ms, 10ms, and 20ms intervals. I execute 1 billion instructions after warming up the caches with 100 million instructions. Note that I adjust the DVFS algorithm parameters (e.g., the threshold values for voltage/frequency changes) so that the processor spends the same balance of time in each frequency/voltage state as in the DVFS profiles collected from the commercial system. My intention in doing this was to analyze the performance impact of disabling LLC ways implemented with small cells whenever the voltage/frequency is reduced. Also, after I obtain the DVFS profile from the baseline system, I applied the same profile to the processor with my proposed LLC architecture at each interval. This gives a fair basis for performance comparisons.

To evaluate the power or energy consumption impact of the proposed LLC architectures, I need to estimate the LLC leakage power dissipation, the dynamic energy consumption of LLC and off-chip memory per access, and the core power consumption. First, the leakage power of each cell type is

measured using HSPICE and 32nm PTM at 60°C. For the conservative estimation, I assume that all live arrays are in sleep mode that retains their states and a subset of the arrays is woken up on demand for every access [73]. This provides a significant reduction in the leakage power consumption of the LLCs. Thus, the LLC leakage power consumption only in sleep mode is used to evaluate the leakage power reduction of the proposed LLC architecture.

The measurement exhibits that C1~C6 cells dissipate 0.342W~0.347W in sleep mode, resulting in ~23W for the LLC leakage power consumption. Second, the dynamic energy consumption per LLC access is estimated using CACTI 6.5 [79]. It estimates that 0.431nJ is consumed per access for the following LLC design parameters: cache size = 8MB, block size = 64, associativity = 16, the number of read/write ports = 1. Off-chip memory, i.e., DDR3 energy consumption per access is 48.4nJ estimated using [80] assuming that the bus width between the processor and off-chip memory is 64 bytes. Note that the DDR3 power consists of three components: background, active power, and read/writeterm power. In the estimation, I only account for the active and read/write power consumed by extra read/write accesses since the background power is always consumed whether it is accessed or not. The extra processor core power consumption due to longer runtime is estimated based on the thermal design power for a commercial high-performance processor, 120W minus the LLC power consumption at 0.9V/2.4GHz. Then, I scale the power consumption of the cores accordingly for different voltage/frequency states. Note that the total energy consumption is obtained by multiplying the power consumption with the corresponding runtime at each voltage/frequency state.

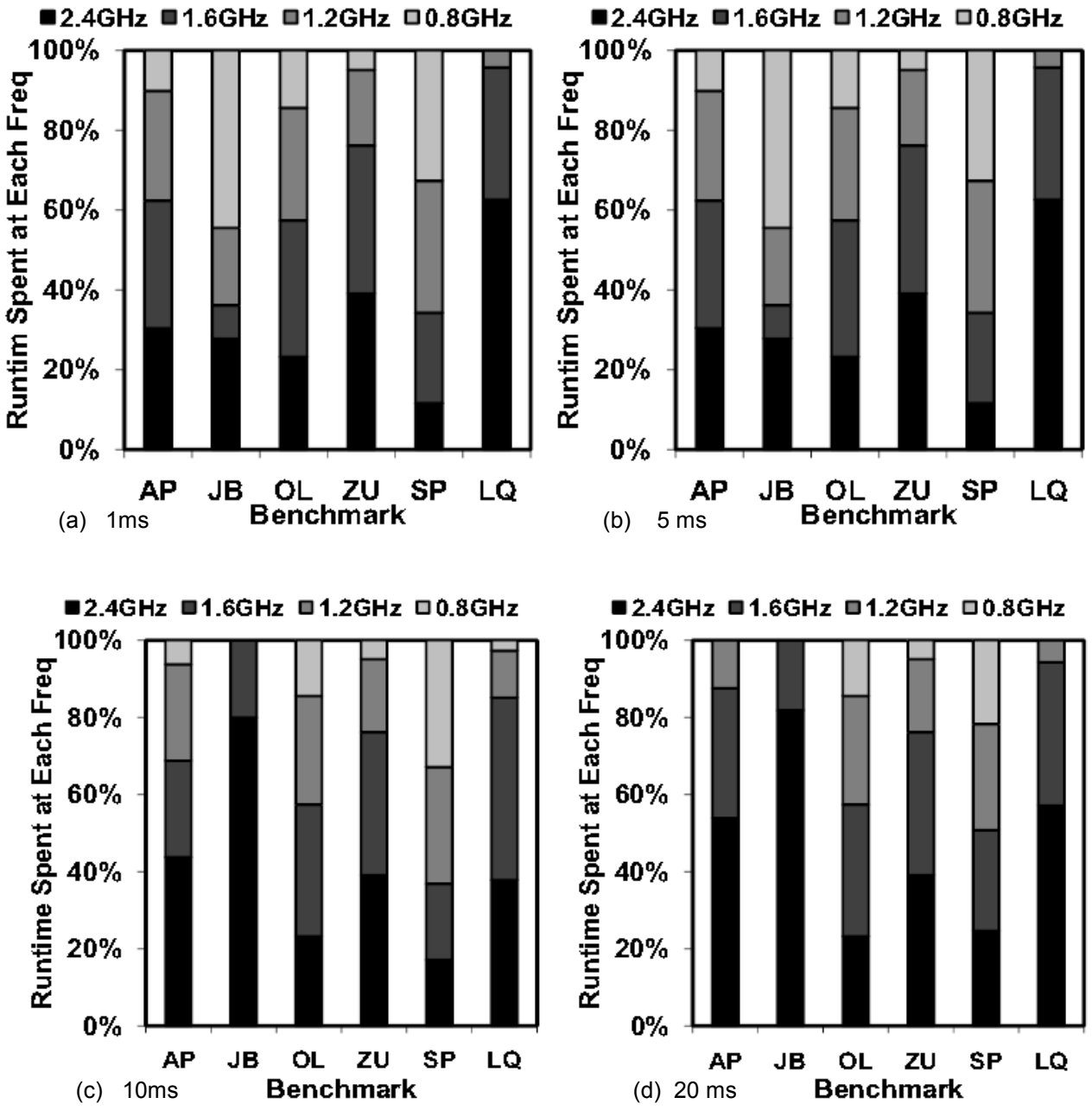


Figure 4-6: Time spent at each frequency/voltage state versus DVFS interval time.

The processor power excluding the LLC leakage power is 100W, 53W, 46W, and 40W at 0.9V/2.4GHz, 0.8V/1.6GHz, 0.75V/1.2GHz, and 0.7V/0.8GHz, respectively. Note that the estimation of the overall processor power is conservative. In fact, it is unfavorable to my proposed LLC

architectures. This is because I assume the highest core power consumption for each frequency/voltage state, which increases total core power consumption most notably for the longer runtime of the proposed architecture.

4.4.2 Simulation Results

Figure 4-6 shows the percentage of time the processor spends at each frequency state based on the DVS algorithm for 1ms, 5ms, 10ms, and 20ms intervals for Apache (AP), JBB (JB), OLTP (OL), Zeus (ZU), sphinx (SP), and libquantum (LQ). The corresponding voltages for 2.4GHz, 1.6GHz, 1.2GHz, and 0.8GHz are 0.9V, 0.8V, 0.75V, and 0.7V, respectively. On average, the 6 workloads spend 35% (44%), 29% (30%), 22% (18%), and 14% (8%) at 2.4GHz/0.9V, 1.6GHz/0.8V, 1.2GHz/0.75V, and0.8GHz/0.7V states when the voltage/frequency change interval is 1ms (20ms). This runtime distribution of the various voltage/frequency states allows the processor to shut down various LLC ways, sufficiently exercising the proposed LLC architectures to determine their performance merit and power impact.

Figure 4-7 and Figure 4-8 plot the runtimes for each of the four different LLC architectures using heterogeneous cell sizes, relative to the conventional architecture at different voltage/frequency change intervals. In this experiment, I observe that the proposed LLC architectures increase the runtime by only 0.49%-1.55%, 0.31%-0.85%, 0.41%-1.13%, and 0.33%-0.94 when the C2M write-back technique is adopted for LLC architectures A, B, C, and D. Note that the runtime of LQ increases as the voltage/frequency change interval increases because the fraction of low voltage/frequency states increases, unlike in other workloads. Further, the C2C write-back technique results in reduced performance loss due to fewer LLC misses and off-chip memory accesses.

On average, the proposed LLC architectures with the C2C write-back technique incur only 0.33%-0.79%, 0.20%-0.42%, 0.23%-0.73%, and 0.22%-0.72% for the LLC architectures A, B, C, and D. This C2C technique shows 23%-49% lower runtime increase than the C2M one. Comparing different LLC architectures, LLC architecture A experiences more frequent shutdowns of the LLC ways and has

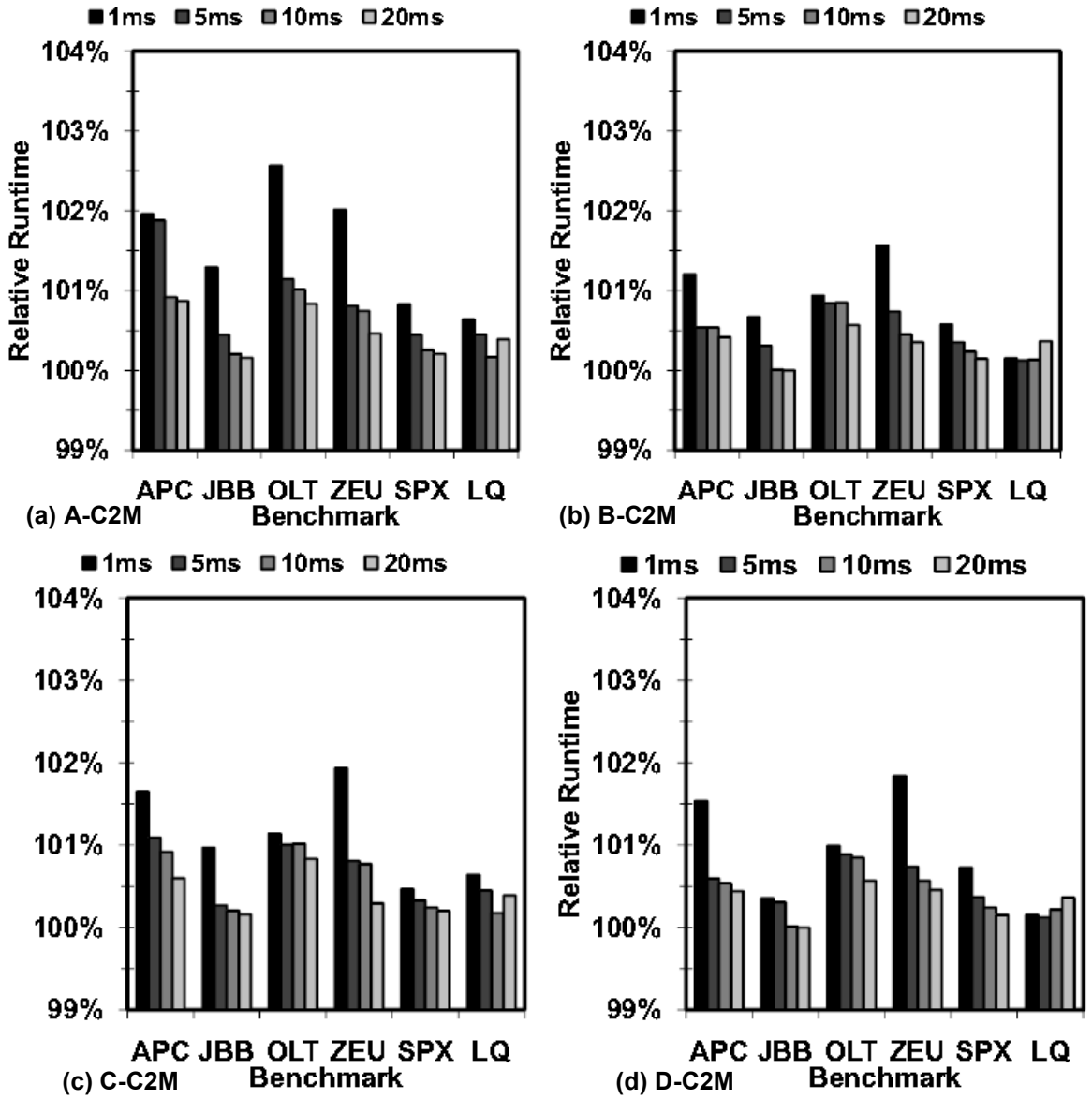


Figure 4-7: Relative runtime versus DVFS interval for different LLC architectures and write-back techniques.

smaller cells than B. Thus, LLC architecture A exhibits higher performance loss than architecture B. Meanwhile, LLC architectures C and D, positioned between A and B in terms of the area, show increases in runtime below A and above B. Finally, these experimental results confirm that the performance loss becomes smaller as the voltage/frequency change interval increases. Note that a short DVFS interval can induce more frequent shutdowns of LLC ways with small cells, generally resulting in more LLC misses and off-chip memory accesses.

Figure 4-9 plots the energy consumption of the proposed LLC architectures, relative to the conventional homogeneous architecture operating at various voltage/frequency change intervals. The energy consumption in Figure 4-9 considers both LLC leakage energy consumption and extra dynamic energy consumption due to the slightly longer runtimes of the processor and additional LLC and off-chip memory accesses. Note that the proposed LLC architectures exhibit less LLC leakage energy consumption both due to the use of smaller cells and to the shutdowns of LLC ways at lower voltage/frequency states. Since the LLC stays in sleep mode at 0.65V during most of runtime and wakes up its small subset briefly for an access, the leakage power does not scale with voltage/frequency unlike the cores. Thus, the relative energy reduction by disabling some LLC ways at low voltage/frequency states outweighs the energy increase due to slightly longer runtime plus additional LLC and off-chip memory accesses. On average, the proposed LLC architecture A, B, C, and D reduce the overall energy consumption by 7%-10%, 5%-7%, 5%-8%, and 6%-9%. For LQ and

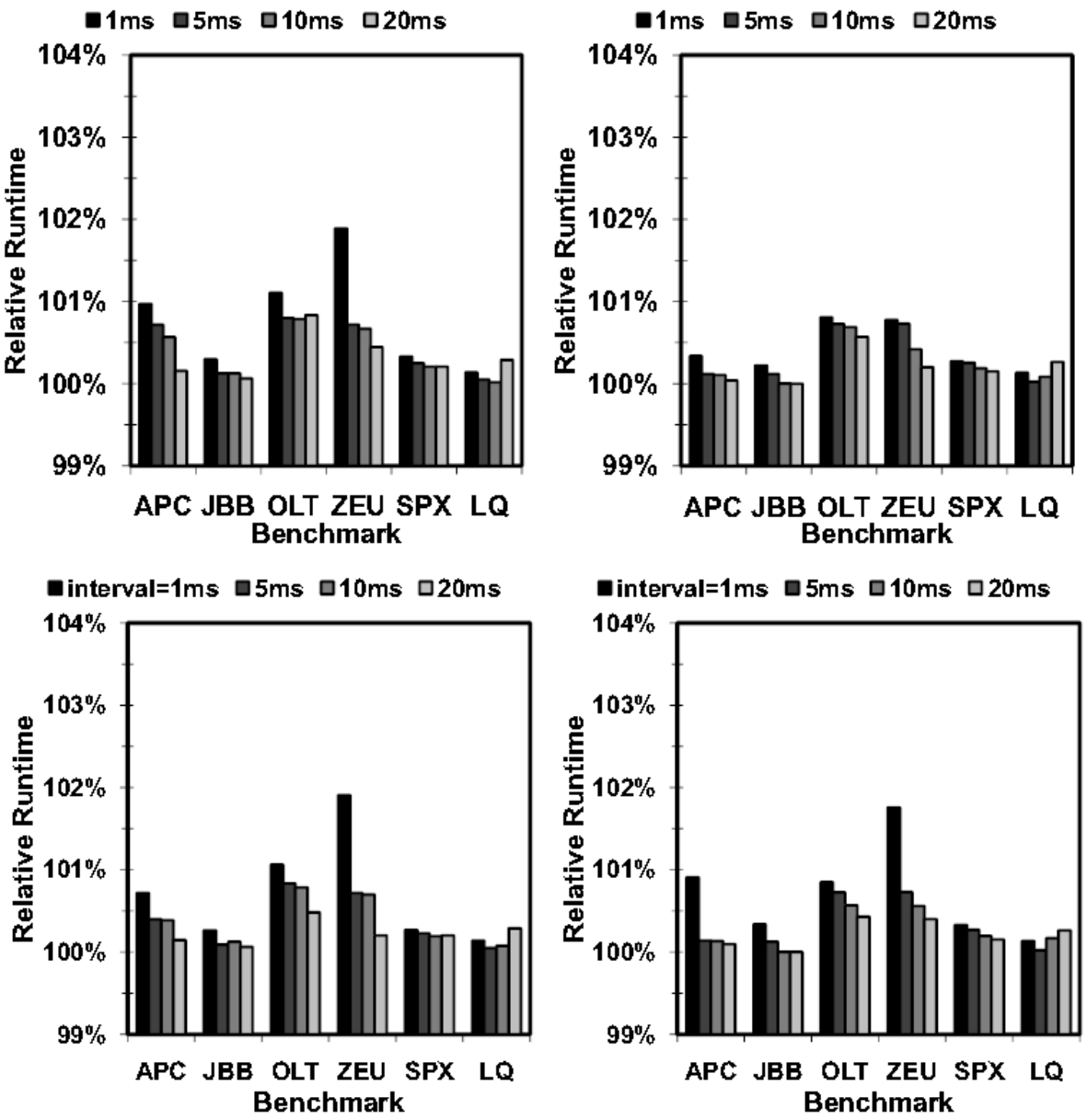


Figure 4-8: Relative runtime versus DVFS interval for different LLC architectures and write-back techniques.

JBB at 20ms, the energy reduction is very small since the processor runs mostly at the highest voltage/frequency state. Note that architectures B and C exhibit less performance loss and, in general, show less energy reduction since a half of the total LLC ways do not shutdown when the voltage is

between 0.9V and 0.8V, thereby losing the leakage power reduction opportunity when the voltage/frequency state is 0.8V/1.6GHz.

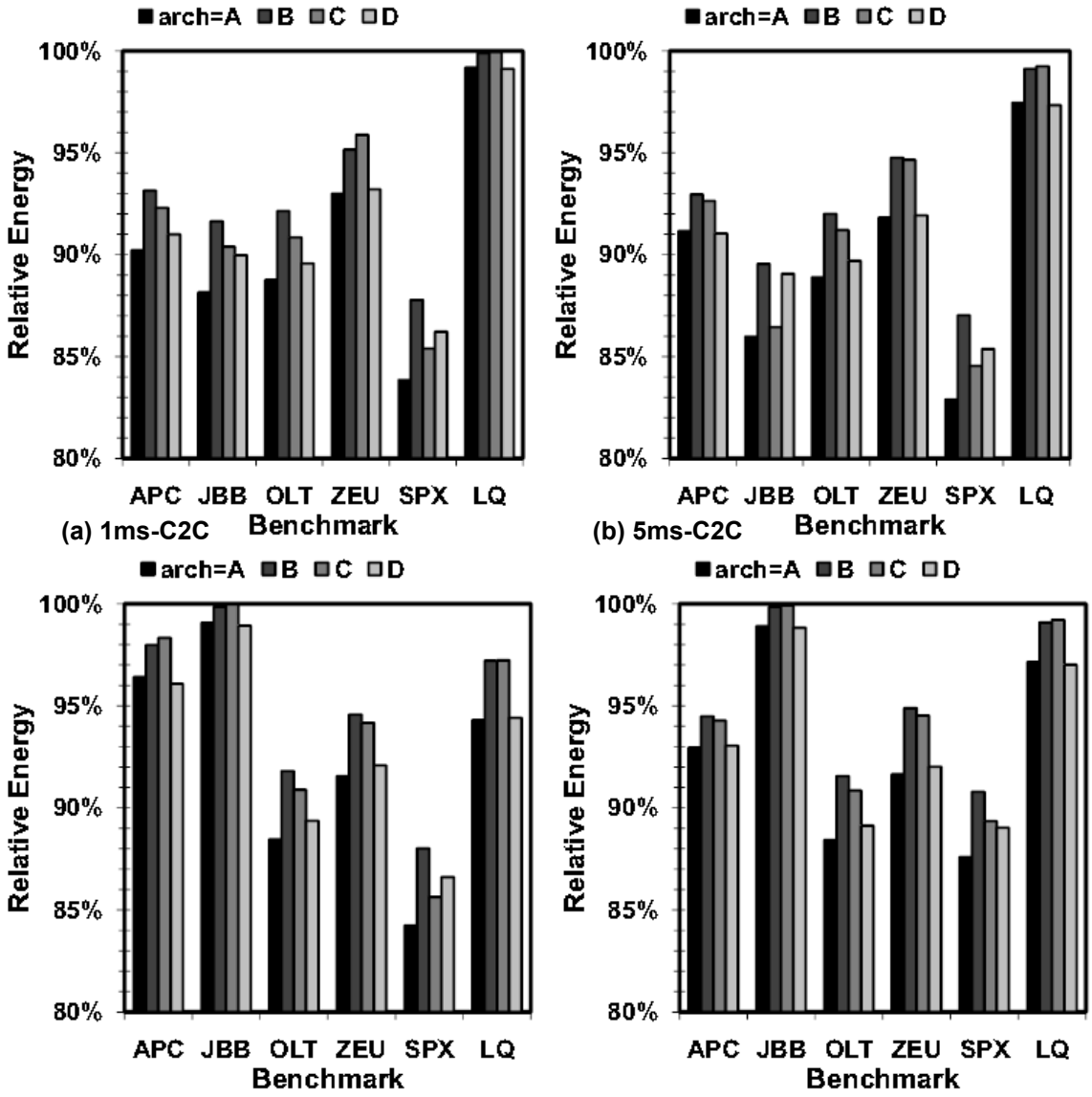


Figure 4-9: Relative energy consumption of the proposed LLC architectures: LLC leakage energy plus extra dynamic energy due to the longer runtime of the processor, and the additional LLC and off-chip memory accesses for different DVFS intervals.

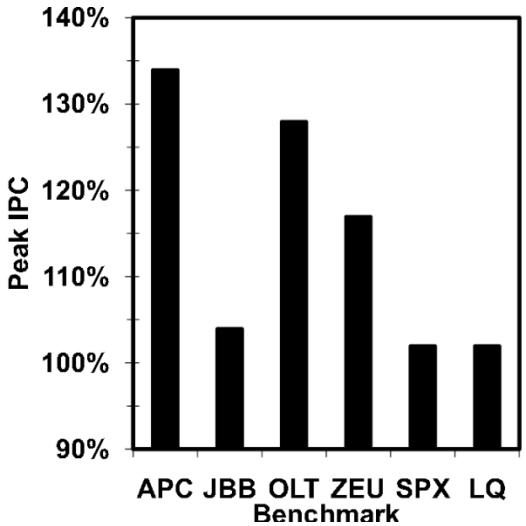


Figure 4-10: Peak IPC comparison between heterogeneous and homogeneous LLC architectures at the same LLC die area.

Figure 4-10 presents the comparison of peak performance between the proposed heterogeneous and conventional homogeneous LLC architectures for the same total LLC area. Since the proposed LLC architectures are 15%-19% smaller than the conventional one, I reduce the capacity of the conventional architecture by 12.75%, i.e., 14 ways instead of 16 ways. For the peak performance comparison, I sample the IPC every 1ms for both LLC architectures setting the processor to run at 0.9V/2.4GHz, and compare the IPC at each interval. Then I choose the values from the interval resulting in the maximum difference. Note that my proposed LLC architectures do not lose any performance at the highest voltage/frequency state since the full LLC capacity is provided. According to this experimental result, the heterogeneous LLC architectures show average 15% higher peak performance than the conventional homogeneous one at the same LLC die area. Finally, the average runtime of the proposed heterogeneous LLC architectures are shorter than that of the conventional homogeneous one analyzing the results shown in Figure 4-1 see the data points with the 75% of 8MB L3 capacity.

4.5 Chapter Summary

In this chapter, I present a cost-effective LLC architecture that uses heterogeneous cell sizes to support both high-performance and low VDDMIN. The proposed architecture exploits the DVFS characteristics of workloads running on high-performance processors, the trade-off between SRAM cell size and VDDMIN, and the lower performance impact of smaller LLC capacity at lower voltage/frequency operating states.

A summary of my results is as follows. First, the proposed LLC architectures reduce the LLC total cell area by 15-20% without impacting performance when in the highest voltage/frequency state. Second, the performance impact of the proposed architectures is negligible when various voltage/frequency states are explored by DVFS as a function of changing performance and power demands. Third, the proposed LLC architectures reduce leakage power since some LLC ways are disabled at low voltage/frequency states. Overall energy consumption is reduced by 5%-10% even though extra energy consumption is required (and accounted for) to support the slightly longer runtimes and more frequent accesses to the LLC and off-chip memory. My proposed LLC architectures also show an average of 15% higher peak performance and shorter runtimes when compared to the conventional homogeneous design for the same die area. These improvements are due to the increased capacity per unit area.

Chapter 5

DRCS: Dynamic Resource and Core Scaling

Although dynamic voltage and frequency scaling (DVFS) has been widely used, the demand to aggressively scale transistor technology and supply voltages will make DVFS window smaller. Eventually, DVFS window may vanish. Thus, architects have been looking for new proposals to alter DVFS technique. Dynamic micro-architecture adaptation techniqes have been a promising solution that leverage power-gating techniques to trade performance and power. In these techniques, micro-architecture resources are dynamically scaled up and down within each core to match application requirements. In a system that dynamic power dominates, these micro-archtectural approaches can be beneficial and simply implementable by gating clock signal of a portion of a resource. However, as transistor technology scales and moves forward, micro-architecture adaptation becomes more challenging due to leakage power increase. In order to address the leakage power increase, we must implement multiple fine-grain voltage domains within each core's resources. In addition, in a multi-core processor with resource and core scaling capability, the scheduling algorithm (i.e., scaling policy) play a critical role to determine the optimal processor configuration. The scaling algorithm can become a complex task as the number of cores and number of scalable resources increases as it needs to perform quickly.

In this chapter, I propose a dynamic resource and core scaling (DRCS) technique as an alternative to DVFS. I envision a power-constrained multi-core processor in which the number of integrated cores

is more than that of operating cores when all the resources of cores are enabled. For such a multi-core processor to maximize performance, I propose (i) joint scaling of the amount of core resources and the number of operating cores (i.e., RCS) and (ii) a runtime system that predicts and adapts the processor accordingly to the best RCS configuration for a given application at runtime.

The remainder of this chapter is organized as follows. Section 5.1 details the experimental methodology. Section 5.2 describes DRCS (Dynamic RCS) under power constraint. Section 5.3 discusses the impact of power constraint. Section 5.4 discusses runtime system. Section 5.5 explains sDRCS (Selective DRCS). Section 5.6 summarizes this chapter.

5.1 Simulation Methodology

For evaluation, I use GEMS [55] full system simulator and McPAT [81] power modeling framework. I configure GEMS similar to a Xeon Tulsa processor; take the corresponding configuration from the McPAT package; and use the 32nm technology for power estimation; see Table 5-1 for the detailed parameters when all the resources are enabled. The processor has 12 cores, but it can operate only 8 cores with full resources enabled due to power constraint.

Table 5-1: Key parameters of a 12-core processor with the full core resources.

# of cores	12	NoC topology	crossbar switch
core fetch/issue/retire	4/4/4	core	OoO
IL1/DL1	private 32KB/32KB	ALU/Multiplier/FPU	4/2/4
ITLB/DTLB	32/32 entries	LSQ	32/32 entries
BTB	2K	integer reg. file/float reg.	128/128
branch predictor	YAGS: 20-15-15	load replay penalty	10 cycles
branch mis-pred. penalty	11 cycle	ROB/IQ	64/32 entries
L2/MSHR	shared 8MB/16 entries	technology	32nm
Cache Coherency Proto-	directory-based MESI	V/F	0.9V /3.2GHz

I use 11 benchmarks: JBB, OLTP, APACHE, and ZEUS from commercial workloads [82]; WUPWISE, MGRID, APPLU, and EQUAKE from SPEC-OMP [83], and SWAPTIONS, BODYTRACK, and BLACKSHOLES from PARSEC [84] (denoted by JBB, OLTP, APCH, ZEUS, WPWS, MGRD, APLU, EQK, SWSP, BDTK, and BKSL). Depending on the degree of contentions related to synchronizations, each thread often executes a different number of instructions [35]. Thus, for fair and consistent comparisons across configurations and intervals, I ensure that each configuration executes the same number of transactions instead of instructions; a transaction typically corresponds to an iteration of a global loop in each executed thread. I run a benchmark for 40 intervals comprised of 2- to 4-billion instructions; each interval is equivalent to tens to hundreds of ms in real machine's time. Although the software threads are scheduled by the OS and executed with fewer cores (after scaling the number of cores) in a time-multiplexed manner, negative performance impact of load imbalancing is small (and reflected in ths simulation) since these applications have far more software threads than 12 cores [85, 86].

5.2 DRCS under Power Constraint

To exploit instruction-level parallelism (ILP) in applications, a processor core is designed to execute multiple in-flight instructions in an out-of-order (OoO) fashion. The more resources (e.g., larger L1/L2 caches and more execution units (EUs)) a core has, the higher performance is for applications with high ILP. Similarly, the more cores a processor has, the more threads it can concurrently execute for applications with high thread-level parallelism (TLP), leading to higher overall performance. However, due to a power constraint, we cannot increase both the amount of resources per core and the number of operating cores at the same time. Consequently, we often have to

trade the amount of resources per core with the number of integrated cores at design time. Moreover, I uniformly scale chosen resources of each core, coupled with the number of operating cores such that the processor does not exceed the chip power constraint for the worst-case power consumption. Hence, the number of feasible RCS configurations is limited to a small number (i.e., three in this study), making it more tractable for implementing a runtime system.

As the prior studies observed, many execution phases of applications often underutilize given core resources due to various reasons (e.g., limited ILP and small memory footprint) [26, 27, 29, 28]. In such a case, disabling some of core resources such as L1/L2 caches and/or EUs may not notably impact single-thread performance, while it can considerably reduce the maximum power consumption

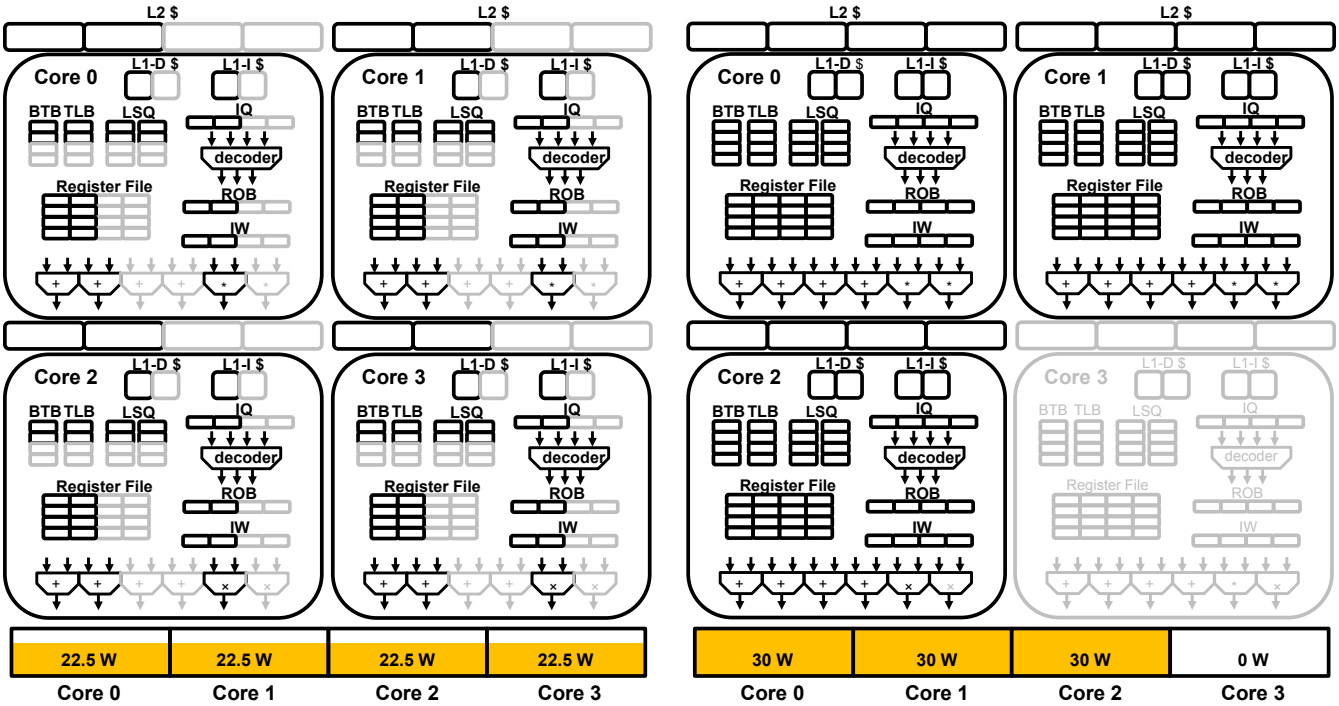


Figure 5-1: A hypothetical example of RCS under a power constraint (90W) for a quad-core processor: (a) all four cores are active with a half of resources disabled (22.5W per core) (b) only three cores are active with full resources enabled (30W per core). The disabled resources are represented with the gray color.

of each core. This in turn allows us to operate more cores (and thus increase the performance of applications with high TLP) under the power constraint. In contrast, if the applications exhibit high ILP but low TLP, disabling some cores but enabling more resources per operating core can lead to higher performance.

Figure 5-1 illustrates a hypothetical example of RCS for a quad-core processor. Assume that the maximum power consumption of the processor is limited to 90W; in this example, I only consider the power consumption of cores after excluding the power consumption by shared components such as memory controllers and network-on-chip (NoC) interconnect, etc. When all the resources of each core are enabled, the maximum power consumption of each core operating at 3.2GHz/0.9V is 30W. Therefore, only three cores can operate due to the power constraint. However, when the amount of resources of some architectural components such as L2/L1 caches, instruction queue (IQ), load-store queue (LSQ), and EUs is reduced by half, the maximum power consumption of each core can be reduced from 30W to 22.5W at the same voltage/frequency (V/F). This in turn allows us to operate all four cores.

Figure 5-2 shows the normalized execution time (ET) of three different RCS configurations that

Table 5-2: Key parameters of three RCS configurations consuming approximately the same maximum power.

Processor Configuration	8-core	10-core	12-core
L2(MB)	8	6	4
IL1(KB)/DL1(KB)	32/32	32/32	16/16
TLB	32	32	16
ALU/Complex ALU/FPU	4/2/4	2/1/2	1/1/1
LSQ	32/32	32/32	16/16
ROB/IQ	64/32	64/32	32/16
YAG Br. Predictor	20/15/15	20/15/15	10/8/8

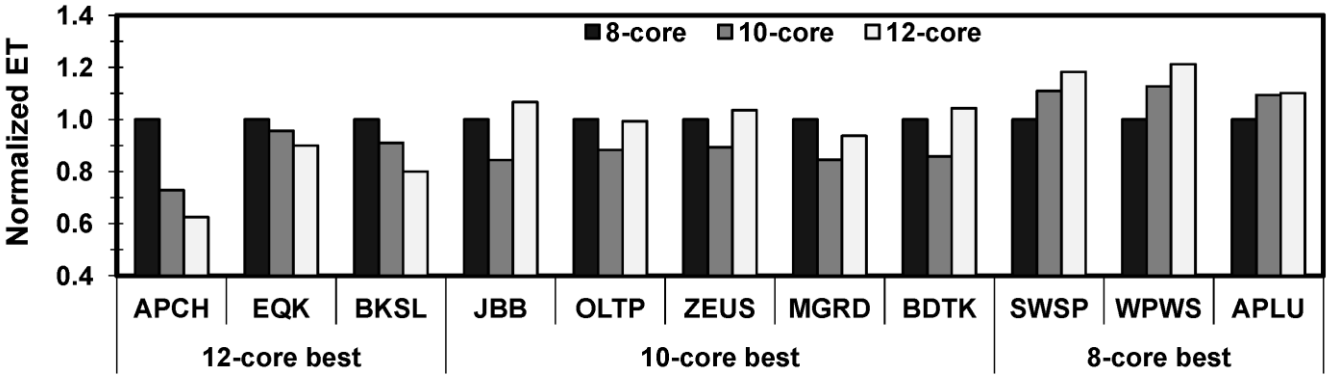


Figure 5-2: Execution time (ET) comparison of 8-, 10-, and 12-core configurations that consume approximately the same maximum power. The ET of each application is normalized to that of the 8-core configuration.

simultaneously vary the amount of core resources and the number of operating cores such that all three configurations consume approximately the same maximum power at 0.9V/3.2GHz. The ET of each configuration is normalized to that of the 8-core configuration. The key parameters of three RCS configurations are listed in Table 5-2. I will discuss how the maximum power consumption of each architectural component scales as the amount of core resources scales in Section 5.3.

I determine these RCS configurations after performing many experiments to analyze the impact of core resource scaling on performance and maximum power consumption of the processor. Then I prioritize the architectural components for resource scalingnce based on (i) the changes of power efficiency (i.e., performance divided by power) after scaling the resources and (ii) the complexity of implementing resource scaling. I choose to apply the resource scaling only for the architectural components that exhibit relatively large power reduction with small performance degradation. Table 5-7 in Section 5.5 presents the changes of power efficiency for major architectural components after reducing the amount of resources by half to motivate my sRCS technique. For instance, I do not reduce the size of physical register-file (PRF) in all three RCS configurations, because reducing the size of PRF decreases considerably more performance than maximum power consumption. Similarly, I reduce

the size of the TLB, LSQ, ROB/IQ, and branch predictor only for the 12-core configuration, as shown in Table 5-2, because they can exhibit notable performance degradation compared to maximum power reduction. Finally, I can vary the fetch width of cores as a category of resource scaling technique, but I do not explore it in this study.

Figure 5-2 clearly demonstrates that the best RCS configuration, which offers the highest performance, varies from benchmark to benchmark. For example, the 8-core configuration provides the highest performance for APPLU, WUPWISE, and SWAPTIONS, while the 12-core configuration gives the highest performance for APACHE, EQUAKE, and BLACKSHOLES; the 12-core configuration offers 31% higher geo-mean performance than the 8-core configuration for these benchmarks. In other words, the amount of resources per core is more critical for APPLU, WUPWISE, and SWAPTIONS, which exhibit high ILP but low TLP and/or have large memory footprints, to achieve the highest performance under the power constraint. On the other hand, the number of operating cores is more critical for APACHE, EQUAKE, and BLACKSHOLES, which generally exhibit high TLP but low ILP and/or small memory footprints, to achieve the highest performance. The performance of the 10-core configuration is the highest for JBB, OLTP, ZEUS, MGRD, and BDTK, indicating that the amount of resources per core and the number of operating core should be balanced to achieve the highest performance. Overall, choosing the best RCS configurations for the benchmarks that benefit from the 10- and 12-core configurations can provide 21% higher geo-mean performance than the 8-core configuration.

5.3 Impact of RCS on Power Consumption

In Section 5.2, I demonstrated that the best RCS configuration leading to the highest performance differs per application depending on the requirements of each application. In modern processors, key architectural components such as on-chip caches, buffers, and queues are designed in a modular fashion so that their size or width can be easily adapted for derivative architectures targeting various market segments (i.e., mobile, desktop, and server). These on-chip memory structures are typically comprised of multiple arrays and each array is equipped with a local power-gating device that can turned it on/off independently [73]. The number of operating EUs can be varied by fine-grained power-gating, which has been explored by the industry due to the growing need for supporting aggressive power management techniques [87]. In general, I share the same (i) hardware mechanism to scale the resource of each architectural component and (ii) associated overhead with [26, 27, 29]. In addition, the number of operating cores is scaled up/down by turning on/off per-core power-gating devices, which are supported by most commercial multi-core processors [88, 89].

Table 5-3 tabulates both the absolute and percentage power consumption of key architectural components of all three RCS configurations. The power consumption of the 8-, 10-, and 12-core

Table 5-3: Breakdown of peak power consumption of cores for different processor configurations.

Processor Configuration		L2 + Dir	IL1 + ITLB + BTB	DL1 + DTLB + LSQ	IQ + ROB	INT	FPU	Others	Core Total
8-core	Power	4.91W	0.43W	0.86W	0.87W	2.88W	2.12W	11.00W	23.07W
	% in Core	21.3%	1.8%	3.7%	3.8%	12.5%	9.2%	47.7%	100%
10-core	Total	3.22W	0.43W	0.86W	0.87W	1.44W	1.06W	10.59W	18.46W
	% in Core	17.4%	2.3%	4.7%	4.7%	7.8%	5.7%	57.4%	100%
12-core	Total	2.01W	0.31W	0.69W	0.69W	0.98W	0.53W	10.23W	15.44W
	% in Core	13.0%	2.0%	4.5%	4.5%	6.4%	3.4%	66.2%	100%

configurations is approximately the same maximum power or thermal design power (TDP) = ~185W; the TDP of an Intel's dual-core Xeon processor manufactured with 65nm technology is 150W [90]and an Intel's 6-core processor manufactured with 32nm technology and operates at a similar frequency is 130W [91]. The power consumption of each core in 8-, 10-, and 12-core configurations is 23.07W, 18.46W, and 15.44W, respectively. The core power consumption number for each configuration includes each core's share of the power consumption of architectural components shared by all the cores, such as memory controllers, NoC, etc. The columns denoted by "L2 + Dir," "IL1+ITLB+BTB," "DL1+DTLB+LSQ," "INT," "FP," and "Others" show the maximum power consumption and percentage values in maximum power consumption of L2 cache and coherent protocol directory; IL1 cache, ITLB, BTB, and branch predictor; DL1 cache, DTLB, and LSQ; integer ALUs, multipliers, and dividers; floating-point adder, multipliers, and dividers; and all other components. In the following section, I discuss how the maximum power consumption of architectural components is calculated after resource scaling.

Set-associative memory components: The first group of memory components is set-associative memories such as L2 (and its coherent protocol directory), IL1/DL1, and BTB; dynamic memory resizing techniques have been widely used for commercial processors to reduce leakage power consumption at runtime [74]. It is possible to shut down a subset of arrays that constitute either ways or sets (depends on cache designs). Although disabling some arrays can reduce the leakage power consumption of the memory components, the dynamic power consumption remains the same. This is because it does not reduce the switching capacitance of accessed arrays; only a subset of arrays in large memory components is accessed regardless of the number of arrays. For example, to shut down 2MB of an 8MB 8-way cache I can shut down ¼ of arrays constituting all the available ways or sets without

impacting the number of parallel accesses to the cache. Scaling the size of L2 to 6MB and 4MB (and the associated coherent protocol directories) reduces the maximum power consumption of the 10- and 12-core configurations by 16.97W ((4.91W − 3.22W) × 10) and 35.85W ((4.91W − 2.01W) × 12), respectively, while scaling the size of IL1/DL1 to 16KB reduces the maximum power consumption of the 12-core configuration by 1.68W ((0.94W − 0.80W) × 12).

Fully-associative memory components: The second group of memory components is fully-associative memories such as ROB, DTLB, ITLB, IQ, and LSQ. These components usually are designed using content-addressable memory (CAM) and some combinational circuits such as comparators and multiplexers. Like set-associative memory, it is possible to shut down a subset of total entries and the associated comparators to reduce their leakage power consumption without impacting the critical path delay [92]. However, this does not reduce the capacitance of tag matching buses that are connected to all the comparators including the disabled ones. Note that a large fully-associative memory component can be designed with multiple CAM arrays that are connected in a hierarchical way. In such a case, disabling a subset of CAM arrays can also reduce dynamic power consumption, but I assume that disabling some entries in the fully-associative memory components only reduces leakage power consumption for a conservative estimation in this study. For instance, disabling a half of the total entries in ITLB, DTLB, BTB, LSQ, and ROB can reduce the maximum power consumption of the 12-core configuration by 3.25W ((1.12W − 0.85W) × 12). In summary, scaling the size of memory components for the 10- and 12-core configurations can reduce the maximum power consumption of the processor by 16.97W and 40.38W, respectively; as indicated in Table 5-2, only L2 cache size is reduced in the 10-core configuration. This is in turn exploited to increase the number of operating cores (with less on-chip memory resources).

EUs: The impact of scaling the number of ALUs, complex ALUs (multipliers and dividers), and FPUs on power consumption is very straightforward. Disabling a subset of such units using fine-grain power- and clock-gating techniques can reduce both dynamic and leakage power consumption. However, this does not reduce the power consumption of the shared operand and result buses that are connected to the inputs and outputs of these units. Scaling the number of EUs can reduce the maximum power consumption of the processor by 24.97W ((4.99W − 2.50W) × 10) and 41.80W ((4.99W − 1.51W) × 12) for the 10- and 12-core configurations, respectively. This power reduction is substantial enough to operate one or more cores under the power constraint. Note that the instruction issue mechanism must be aware of disabled EUs and it should not issue instructions to the disabled EUs.

Others: In this study, I do not apply the resource scaling techniques for the remaining architectural components, although it is possible to scale some of them, such as the number of memory controllers, the width of fetch, decode, and issue stages, and the size of PRF.

Putting it together: McPAT can provide the breakdown of maximum dynamic and leakage power consumption of each component in a processor. I begin with the estimation of the maximum power consumption of a 12-core processor with the full resources per core to accurately consider the power consumption of all the shared resources (e.g., on-chip interconnect, memory controllers, etc.) and extract dynamic and leakage power consumption of each component including its subcomponents (e.g., arrays, decoders, etc. in L1/L2). Then, I subtract the dynamic and/or leakage power consumption of disabled subcomponents based on the methodology discussed in this section. Scaling the chosen resources of each core reduces the maximum power consumption of each core by 4.61W and 7.63W for the 10- and 12-core configurations, as summarized in Table 5-3. Such power reduction can provide

enough power headroom to operate two and four more cores for the 10- and 12-core configurations than for the 8-core configuration since each core consumes approximately 18.5W and 15.5W, respectively.

5.4 Runtime System

In this section, I propose a runtime system (i.e., a scaling policy) that can determine (i) the best RCS configuration for a given application and (ii) adapt the processor accordingly. Then, I evaluate its effectiveness and discuss its runtime overhead.

5.4.1 Performance Comparison at Each Interval

Figure 5-3 plots the ETs of 8-, 10-, and 12-core configurations that run BLACKSHOLES and ZEUS, at each runtime interval. The ETs of the 10- and 12-core configurations are normalized to that of the 8-core configuration at each interval. Each interval in a benchmark is comprised of the same number of transactions to perform the same amount of work, typically corresponding to 60- to 150-million instructions (i.e, tens to hundreds of ms) depending on a benchmark and its interval.

While a particular RCS configuration always outperforms two other RCS configurations across all the intervals for APACHE, SWAPTIONS, EQUAKE, and BLACKSHOLES, the RCS configuration leading to the highest performance changes over intervals for ZEUS, OLTP, JBB, MGRID, APPLU, WUPWISE, and BODYTRACK. For instance, the 12-core configuration always shows higher performance than the 8- and 10-core configurations across all the runtime intervals for

BLACKSHOLES. On the other hand, either the 8- or 10-core configuration can provide the highest performance at some runtime intervals for ZEUS.

Assume that I can oracularly choose the best RCS configuration that can lead to the highest performance at each interval (i.e., the best dynamic RCS) and compare its overall performance with the best performance that can be achieved by applying only one of the RCS configurations across all the intervals for each application (i.e., the best static RCS). Among the applications that benefit from the dynamic RCS, as tabulated in Table 5-4, the dynamic RCS can provide notably higher performance only for BODYTRACK and ZEUS than the best static RCS, while offering only negligibly higher performance for JBB, OLTP, MGRID, and WUPWISE.

5.4.2 Best Configuration Predictor

Considering small performance improvements using the dynamic RCS over the best static RCS, I focus on developing a predictor that can determine the best (static) RCS configuration at runtime.

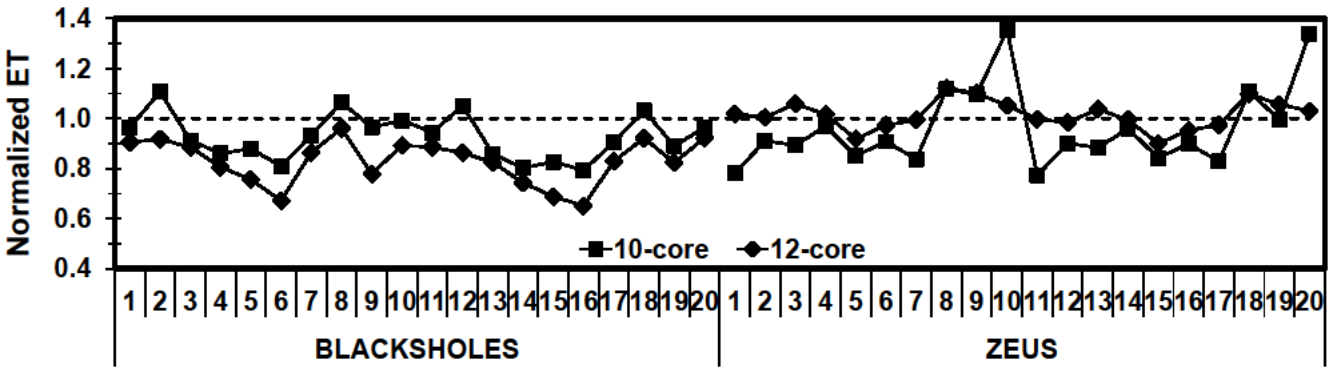


Figure 5-3: ET versus runtime interval of the 8-, 10-, and 12-core configurations. The performance of each application is normalized to that of the 8-core configuration at each runtime interval.

Table 5-4: Performance improvement with dynamic over static configurations.

Applications	JBB	ZEUS	OLTP	MGRD	WPWS	BDTK
Performance Improvement (%)	0.8%	4%	0.5%	1.6%	0.1%	31%

Need for using an ML approach: A simple performance sampling technique can be sufficient, if we can repeat the execution of the same interval for all three configurations, which is non-trivial and costly considering the duration of intervals. We cannot just compare performance of configuration A at interval i with that of B at interval i+1 to determine the best configuration as the performance reference points are different at interval i and i+1. Consequently, determining the best configuration at runtime requires considering numerous performance counters, leading to a complex multi-dimensional classification problem in which a machine learning (ML) approach can be very effective.

Predictor architecture: To determine the best RCS configuration, I take support vector machine (SVM) that is a supervised learning model and widely used for classification and regression analysis. An SVM takes a set of inputs (i.e., performance counter values) and predicts which of two possible classes forms the output (i.e., the best RCS configuration) as a non-probabilistic binary linear classifier. For example, an SVM predicts whether or not a particular RCS configuration (e.g., 12-core configuration) is the best for given counter values. Hence, a simple predictor can be comprised of M SVMs where M is the number of feasible RCS configurations (= 3 in my study).

Figure 5-4 shows my proposed predictor architecture comprised of 3 sub-predictors (SVM_8, SVM_{10}, and SVM_{12}), each of which consists of 3 SVMs, and how it determines the best RCS configuration. Each predictor only receives inputs from the associated RCS configuration. For

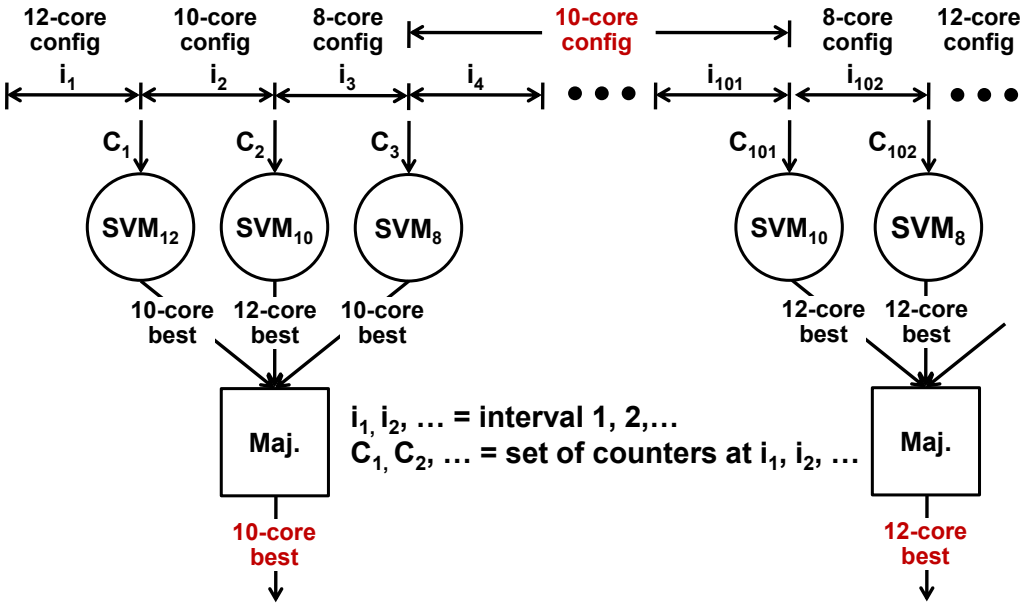


Figure 5-4: Proposed predictor architecture, comprised of 3 sub-predictors based on SVMs, to determine the best RCS configuration for *n* = 1.

instance, SVM_{10} only receives inputs generated by the 10-core configuration and predicts whether or not the best RCS configuration is the 8-, 10, or 12-core configuration; see "Predictor inputs" to justify why I split the predictor into three sub-predictors.

The predictor explores all three feasible RCS configurations over 3 × n consecutive intervals, starting with the 12-core configuration; see Section 5.4.3 for the justification of starting with the 12-core configuration. At the end of each n consecutive intervals, the sub-predictor, which is associated with the current RCS configuration, predicts the best RCS configuration based on the collected counter values during the intervals. At the end of the third n consecutive intervals, the predictor first decides the best RCS configuration based on the majority rule; in Figure 5-4 where n = 1, SVM_8 and SVM_{12} predict that the 10-core configuration is the best while SVM_{10} predicts that the 12-core configuration is the best. Thus, the predictor decides that the 10-core configuration is the best.

When all three sub-predictors disagrees, the predictor chooses the best configuration based on the sub-predictor with the highest confidence (i.e., the longest distance to the SVM's hyper-plane). The runtime system then takes necessary actions to apply the decided configuration to the processor; see Section 5.4.3 for the necessary actions and their overhead to change the processor configuration. Finally, it maintains the decided RCS configuration for a pre-determined period of time (e.g., 100 intervals) as illustrated in Figure 5-4 and/or until it detects a substantial change of given application's execution phase (e.g., [93]).

Predictor inputs: Table 5-5 categorizes the counters, taken by the best-configuration predictor, into three groups. The first "System Performance" group includes each core's IPC and ET to evaluate the performance benefit of the current configuration. The second "Memory Resource" group consists of access and miss rates of each core's on-chip caches and bandwidth utilization of each off-chip DRAM channel. These counters indicate whether or not a given application is memory-intensive (i.e., the on-chip cache sizes and/or off-chip DRAM bandwidth is sufficient for a given application); for some applications, increasing the number of threads does not improve performance due to limited off-chip DRAM bandwidth [35]. The third "Core Resource" group encompasses utilization and stall rates of ROB, IQ, FPU, and ALU as well as a rate of branch miss-prediction. These counters demonstrate how much the core configuration is matched with the resource requirement of a given application. I consider the utilization counters for SVM_8 since all core resources are fully activated but they may not be fully utilized. On the other hand, I take the stall counters for SVM_{12} because only a small amount of core sources are activated and thus they may cause more stalls due to limited core resources. For SVM_{10} I take both utilization and stall counters because the 10-core configuration is between those two cases. Since I need to consider different counters depending on what is the current RCS configuration,

Table 5-5: Summary of inputs to SVMs.

Category	Counters
System Performance	IPC and ET
Memory Resources	access/miss rate of (L2, L1D, L1I) and bandwidth utilization of off-chip DRAM channel
Core Resources	utilization/stall of (ROB, IQ, ALU, FPU, LSQ) and branch prediction accuracy

I split the predictor into 3 sub-predictors, each of which is customized for each RCS configuration, leading to higher accuracy according to my preliminary evaluations.

Predictor training: I take an off-line training method in this study due to long cycle-level simulation time for evaluations involving an on-line training approach. For an off-line training method, I collect 40 sets of chosen counter values from 40 execution intervals of a benchmark running on a particular RCS configuration; the number of sets per configuration per benchmark is limited by my simulation time constraint but it can be far more with real hardware. I also obtain 2×40 sets by moving-averaging counter values over 2 and 4 intervals, respectively. For three feasible RCS configurations per benchmark, I have total 3×120 (=360) sets and they will be associated with and trained to predict the best RCS configuration of the benchmark. Then L × 120 sets collected from L benchmarks running on a particular RCS configuration are applied to sub-predictor associated with the RCS configuration for the training. For example, I apply 10 × 120 sets from 10 benchmark executed on the 8-core configuration to SVM_8 that is comprised of 3 outputs (i.e., "8-core-best," "10-core-best," and "12-core-best"). Among the 10 × 120 sets, the 3 × 120 sets from APACHE, EQUAKE, and BLACKSHOLES and 3 × 120 sets from APPLU, WUPWISE, and SWAPTIONS will be trained to predict "12-core-best" and "8-core-best," respectively, during the training phase of SVM_8. The remaining sets from the rest of the benchmark will be trained to predict "10-core-best." This process is

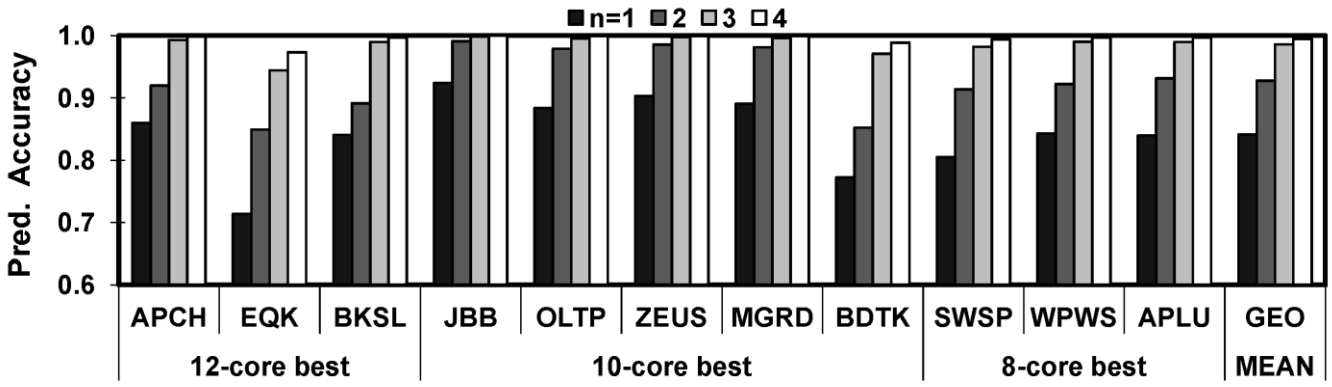


Figure 5-5: Prediction accuracy for *n* = 1, 2, 3, and 4 intervals.

repeated for SVM_{10} and SVM_{12}. Finally, to evaluate the prediction accuracy for a given benchmark, I exclude all the input sets from the benchmark being evaluated during the off-line training phase.

Predictor accuracy: Figure 5-5 shows the accuracy of the proposed predictor. The geo-mean of the prediction accuracy is 84%, 93%, 99%, ~100% for n = 1, 2, 3, and 4, respectively. To calculate the prediction accuracy for a benchmark, I try all possible combinations of randomly picking 1, 2, 3, and 4 consecutive intervals out of 40 available intervals from each RCS configuration. Most benchmarks show close to 100% prediction accuracy when n is greater than 2. However, some benchmarks such as BODYTRACK and EQUAKE exhibit relatively poor prediction accuracy because of a very wide range of performance variance across intervals. Note that, EQUAKE shows very small performance difference between configurations. Therefore, inaccurate prediction does not impact performance notably. In summary, I can predict the best RCS configuration with well over 90% accuracy by observing only 15% of total intervals, 2 intervals per configuration (= 6 intervals) out of 40 intervals in this experiment.

This proposed predictor can provide very high prediction accuracy to determine the best RCS configuration that can lead to the highest overall performance over a large number of intervals even for

applications in which the best RCS configuration changes over intervals (e.g., ZEUS, OLTP, JBB, MGRID, APPLU, WUPWISE); I can consider that these applications show phase behavior in a time scale of seconds [93]. Considering the simulation time, I limit the number of intervals to 40, which corresponds to a few seconds for most applications. However, some applications show the phase behavior over time scale larger than a few seconds [93]. For such applications, I can periodically apply the prediction technique and adjust the configuration according to capture a large time-scale phase behavior. To this end, Isci et al. proposed a runtime technique to monitor and predict the runtime phases of applications to increase power efficiency using DVFS [93]. Sherwood et al. propose Simpoint to find a portion of program that is representative of the entire program's execution [94]. Later, Lau et al. propose an on-line technique to predict different phases of applications and resource requirement at runtime [95]. Similar techniques can be adopted to determine whether or not I need to update the best RCS configuration.

5.4.3 Necessary Support and Runtime Overhead

To decide the best RCS configuration, the predictor begins to collect counter values for n (= 2 in this study) consecutive intervals for each configuration in the order of 12-, 10-, and 8-core configurations. In other words, I change the RCS configuration at most three times. This gradually

Table 5-6: Performance improvement of oracular and runtime RCS with *n* = 2. The performance of the 10- and 12-core configurations is normalized to that of 8-core configuration.

	12-core best			10-core best					8-core best		
	APCH	EQK	BKSL	JBB	OLTP	ZEUS	MGRD	BDTK	SWSP	WPWS	APLU
Oracle	60.0%	11.1%	25.0%	18.5%	13.2%	11.9%	18.3%	16.6%	0.0%	0.0%	0.0%
Runtime	54.4%	10.1%	22.6%	15.7%	11.8%	10.4%	16.7%	14.3%	-1.4%	-1.7%	-1.1%
Runtime/Oracle	90.1%	91.2%	90.3%	84.9%	88.9%	87.0%	91.5%	86.1%	98.6%	98.7%	99.3%

turns on more core resources for fewer operating cores. Therefore, scaling down the core resources (i.e., reducing the size of on-chip memory components and flushing core pipelines) occurs at most once (i.e., from the 8-core configuration back to either 10- or 12-core configuration) during a substantial period of runtime. Then I maintain the best RCS configuration determined by the predictor until I observe a considerable phase change from a given application, which may occur every a few seconds [93]. Thus, I expect that the overall overhead of runtime RCS should be very small; see Table 5-6 for runtime overhead. The key hardware mechanisms to support runtime RCS are already available in the power control unit (PCU), a microprocessor, which monitors performance and power consumption of each core; dynamically adjusts V/F state; manages the sequences of turning on/off cores and a part or all of on-chip caches, etc., in a multi-core processor [74].

Changing the number of operating cores: Since applications typically have more software threads than 12 cores [85, 86], these many threads are executed with fewer cores in a time-multiplexed manner. Before changing the number of operating cores at runtime, the runtime RCS system notifies the change to the thread scheduler so that it can dispatch an appropriate number of software threads to the cores; I assume the same OS thread-scheduler as proposed by Suleman et al. [35] for changing the number of active threads at runtime and it takes only few μs for switching of threads between cores [96] while one interval in my evaluation is equivalent to tens of ms.

Reducing on-chip shared cache size: When the cache size should be reduced due to a change of the configuration, the cache controller writes back all the dirty cache lines to the main memory and then decreases the L2 cache size. This starts by walking through all the cache lines in the cache section that is going to be shut down. To evict any cache line in “M” state, the cache controller sends the data

to the main memory and updates the directory to “I.” To evict any cache line in “E” state, the cache controller just updates the directory to “I.” To evict any cache line in “S” the cache controller sends an invalidation request for the cache line to the NoC and waits to get invalidation acknowledgements from all the sharers. Then, the controller sends the data to the main memory and updates the directory to “I.” After completing this process, the cache controller reduces the cache size. The performance impact of changing the size of a large shared cache even every 1ms is reported to be less than 0.5% on average [1].

Flushing core pipelines and order of downsizing: When the amount of core resources should be scaled down, first the cores stop fetching new instructions and wait until outstanding misses are serviced. Second, I allow all the previously fetched instructions in the IQs/ROBs to be executed and retired and begin to flush L1 caches in parallel across cores to be disabled. The ROB has 64 entries, so it takes at least 16 cycles for a 4-wide core to retire all the instructions when there is no outstanding cache miss. Finally, while the L2 cache is flushed and downsized, the processor can shut down a portion of the resources of the cores using a fine-grained power-gating technique and lets the instructions be fetched in the new RCS configuration. I assume/model that I halt the core(s) and flush/initialize the pipeline(s) and caches for the components impacted by power-gating on/off during the reconfiguration; the sleep/wake-up latency of fine-grained power-gating just considering the stabilization of power distribution network (PDN) of execution units is in the order of a couple of cycles for ALUs [87] and tens of cycles for FPUs; in my modeling, I assume 10 and 100 cycles for shutting down ALUs and FPUs, respectively.

Predictor computational overhead: The current predictor implementation is based on libsvm containing redundant routines for development and debugging. I measure the computational overhead of the predictor by running the binary in my simulator configured with the parameters shown in Table 5-1. I observe that it takes 0.47ms. 1.53ms, 1.58ms for SVM_8, SVM_{10}, and SVM_{12} to predict the best configuration using a single core (out of 8, 10, 12 available cores). This overhead can be substantially shortened and eliminated by developing optimized SVM code and run it on the PCU.

Overall overhead of runtime RCS: My runtime RCS significantly differs from typical fine-grained dynamic power-gating techniques that turn on/off components based on activity changes of architectural components during a relatively short period of runtime. Note that the sleep/wake-up latency of power-gating just considering the stabilization of PDN is less than tens of ns [97]; most performance penalty is incurred only when the size of on-chip caches is reduced. Furthermore, the overhead of either completing or invalidating in-flight instructions is negligible (takes few hundred cycles), because I need at most one configuration change that requires a shutdown of resources in a long period.

Modeling and evaluating the overhead of runtime RCS, I observe that runtime RCS applied for 40 intervals can deliver 92% of the maximum performance improvement achieved by an oracular RCS (i.e., using the best RCS configuration from the start of executions) on average; Table 5-6 compares the performance improvement of the oracular and runtime RCS, and the performance of 10- and 12-core configurations is normalized to that of 8-core configuration. 9% lower performance than the oracular RCS is resulted by three factors: (i) execution of some intervals with the sub-optimal configurations; (ii) the performance penalty of increasing/decreasing on-chip caches, flushing core pipeline, and

Table 5-7: Impact of resource scaling of each component on power efficiency of the 10-core configuration.

	Half L2s	Half L1s	Half TLBs	Half ALUs	Half FPUs	Half PRF	Half LSQ	Half IQ
ZEUS	0.32%	-2.59%	0.02%	6.53%	4.86%	-5.02%	0.01%	-5.77%
JBB	-5.77%	-2.74%	0.02%	6.13%	4.86%	-6.71%	0.01%	-8.48%
OLTP	-1.44%	-3.40%	0.02%	6.50%	4.86%	-4.48%	0.01%	-6.23%
APACHE	-8.96%	-2.07%	0.02%	3.87%	4.86%	-8.16%	0.01%	-9.46%
EQUAKE	10.70%	-18.53%	0.02%	3.32%	4.33%	-4.11%	0.01%	-3.10%
BODYTRACK	-5.39%	-13.39%	0.02%	-5.62%	4.75%	-3.56%	0.01%	-7.29%

turning on/off cores; and (iii) execution of predictor code. Finally, I limit the number of intervals to 40 due to excessive simulation time, but I expect the runtime RCS to provide higher performance with more intervals, amortizing the cost of (i) and (ii).

5.5 sRCS

Previously, I explored RCS that uniformly scales the chosen resources of each core. In this section, I explore RCS that selectively scale the resources of each core (i.e., sRCS) to better exploit the unique characteristics of memory- and compute-oriented applications.

Motivation and rationale: Table 5-7 shows the impact of scaling the resources of each component on power efficiency (i.e., performance/Watt) of the 10-core configuration. I scale the amount of each architectural component's resource at a time and measure the relative change of performance and maximum power consumption. Reducing the size of IL1, DL1, PRF, and IQ by half notably degrades the power efficiency of all the examined applications. On the contrary, reducing the number of FPUs by half improves power efficiency of all the examined applications. Scaling the L2 size and the number of integer ALUs exhibits varying impacts on power efficiency depending on applications; see the shaded columns in Table 5-7. For instance, the power efficiency of EQUAKE considerably improves with a half of L2 cache size while that of APACHE, JBB and BODYTRACK notably degrades.

Similarly, all applications except for BODYTRACK show improvement in power efficiency when the number of ALUs is reduced by half. For JBB, I can hypothesize that increasing the L2 size while decreasing the number of ALUs can further improve performance under a power constraint.

Performance improvement: For sRCS, I provide four sub-configurations including the original 10-core configuration considered earlier. The number of operating cores is ten for all four sub-configurations, but one sub-configuration (i.e., 10-core memory-oriented configuration) has the full 8MB L2 cache size like the 8-core configuration with fewer ALUs and FPUs like the 12-core configuration. The two other sub-configurations (i.e., 10-core alu- and fpu-oriented configurations) have 4MB L2 cache size like the 12-core configuration but with more ALUs or FPUs like the 8-core configuration. Table 5-8 summarizes these four configurations and highlights the key differences among the sub-configurations from the original 10-core configuration (i.e., 10-core balanced configuration) are shown in the two shaded rows. All four sub-configurations consume approximately the same maximum power (~185W). Note that I cannot provide sRCS for the 8- and 12-core configurations in this study. This is because I only have three configurations; (i) the 8-core configuration uses all the resources of each component while (ii) the 12-core configuration uses a bare

Table 5-8: Key parameters of cores for four 10-core sub-configurations consuming approximately the same maximum power.

Processor Configuration	memory oriented	balanced (original)	alu oriented	fpu-oriented
L2(MB)	8	6	4	4
IL1(KB)/DL1(KB)	32/32			
TLB	32			
ALU/Complex ALU/FPU	2/1/1	2/1/2	4/2/1	1/1/4
LSQ	32/32			
ROB/IQ	64/32			
YAG Br. Predictor	20/15/15			

minimum amount of resources for ALUs and FPUs in each core. Therefore, there is no way to increase (or decrease) the amount of resources of these two configurations. However, with more cores, I expect that more configurations can support sRCS, but I leave it as future work.

Figure 5-6 compares the ETs of four 10-core sub-configurations. The ETs are normalized to the 10-core balanced configuration. The three commercial workloads, JBB, OLTP, and ZEUS that benefit from the 10-core original configuration show further improvement with the 10-core memory-oriented configuration since they exhibit very low ALU and FPU utilization with high cache misses. On the other hand, MGRID and BODYTRACK show further improvement with the 10-core alu- and fpu-oriented configurations, respectively. On average, sRCS can provide 6% performance improvement additional to 11% performance improvement for the applications that exhibit the highest performance with the 10-core baseline original configuration.

Finally, let us consider all the applications that exhibit the highest performance with the 8- and 12-core configurations (APACHE, EQUAKE, BLACKSHOLES, SWAPTIONS, WUPWISE, and

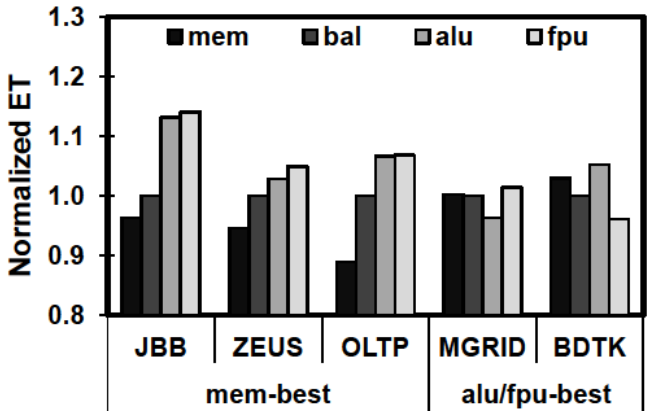


Figure 5-6: ET comparison of four 10-core sub-configurations with SRS. The ETs are normalized to the 10-core balanced configuration.

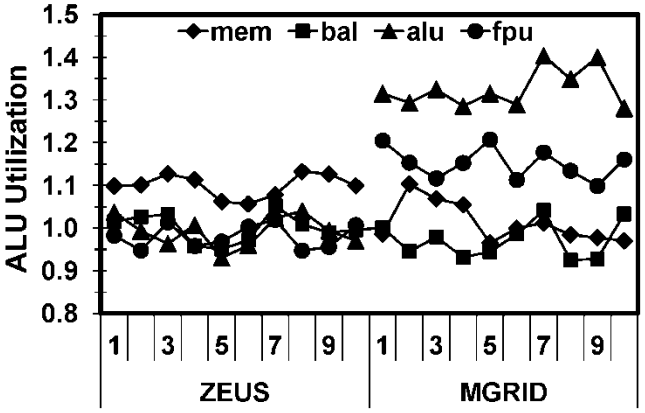


Figure 5-7: ALU utilization versus runtime interval of four 10-core sub-configurations with sRCS. The ALU utilizations are normalized to the 10-core balanced configuration.

APPLU). For these applications, I compare the highest performance with either the 8- or 12-core configuration to that with the best 10-core sub-configuration. I observe that the performance of the best 10-core sub-configuration is never higher than that of the 8- or 12-core configurations for these applications.

Best sub-configuration predictor: The determination of the best 10-core sub-configuration can be performed in two steps. First, the predictor proposed in Section 5.4.2 determines the best configuration out of the original 8-, 10-, and 12-core configurations. Second, if the chosen best RCS configuration can support sRCS (i.e., the 10-core configuration in this study), I determine the best sub-configurations.

Figure 5-7 shows the ALU utilization comparison of four 10-core sub-configurations at each runtime interval for ZEUS and MGRID. First, all the five applications, which exhibit the highest performance with the 10-core balanced configuration, consistently favor one particular 10-core sub-configuration (e.g., the memory- and alu-oriented configurations for ZEUS and MGRID, respectively) and the minimum ALU utilization of the best sub-configuration is almost always higher than the other

Table 5-9: Key parameters and operating points of three 10-core sub-configurations + limited V/F scaling consuming approximately the same maximum power.

Processor Configuration	low-V	balanced (original)	high-V
V/F	2.7GH/z0.85V	3.2GHz/0.9V	3.6GHz/0.95V
L2(MB)	8	6	4
ALU/Complex ALU/FPU	4/2/4	2/1/2	1/1/1

sub-configurations across all the intervals that I examine; I only show 10 intervals for each application in Figure 5-7 and the analysis for the time period substantially longer than 40 intervals is left as future work. In such a case, I can use the ALU utilization to determine the best sub-configuration. This gives almost 100% prediction accuracy for sampling only one interval for each sub-configuration for all the five applications.

Comparison with RCS + Limited V/F Scaling Range: Assuming a limited V/F scaling range can be still available, I compare RCS + limited V/F scaling with sRCS. Table 5-9 tabulates the key configuration parameters and operating points of three 10-core sub-configurations. I can afford larger L2 cache and more EUs at 0.85V at the cost of lower operating frequency (i.e., low-V 10-core configuration) while I can operate all the cores at higher frequency (3.6GHz instead of 3.2GHz) at the expense of smaller L2 cache and fewer EUs at 0.95V (i.e., "high-V" 10-core configuration). Similar to sRCS, I examine only the 10-core configuration because the 8- and 12-core configurations cannot support low- and high-V configurations, respectively; although the 8- and 12-core configurations can support high- and low-V configurations, respective, in this study, I limit my study to the 10-core configuration to contrast this result to sRCS.

Figure 5-8 compares the normalized ET of the low-V and high-V 10-core configurations with the best 10-core configuration using sRCS. I only evaluate the applications exhibiting the highest

performance with the (balanced) 10-core configuration to contrast this result to sRCS. The low-V 10-core configuration provides 6%-19% higher performance than the balanced 10-core configuration while the high-V 10-core configuration offers 2.4% higher performance for OLTP; JBB shows the highest performance with the balanced 10-core configuration. In comparison, ZEUS, JBB and OLTP with the best 10-core sRCS configuration show 4%-10% higher performance while MGRD and BDTK exhibit 8% and 15% lower performance than the best of the two 10-core configurations using limited V/F scaling.

5.6 Chapter Summary

With technology scaling, supporting a sufficient V/F scaling range for effective power management techniques becomes too expensive in terms of either chip area and/or power consumption. Facing such a challenge, I exploit resource scaling as a mean to compensate for a lack of V/F scaling range and propose RCS to improve performance of power-constrained multi-core processors in this chapter. My

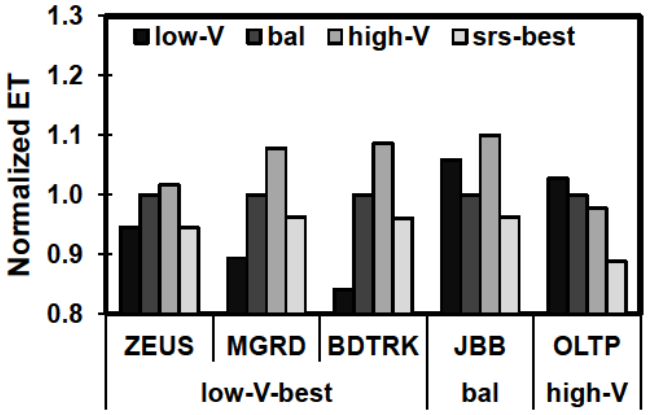


Figure 5-8: ET comparison of 10-core configurations with a limited V/F scaling and the best sRCS. The ETs are normalized to the 10-core balanced configuration. configurations with sRCS. The ETs are normalized to the 10-core balanced configuration.

experimental results demonstrate that RCS can improve a geo-mean performance by 21% over the baseline 8-core configuration. I also propose a runtime system that predicts the best RCS configuration for a given application and adapts the processor accordingly at runtime. The predictor based on an ML algorithm only needs to examine 15% of runtime intervals to achieve accuracy well over 90% and runtime RCS can offer 92% of the maximum performance that can be achieved by the oracular RCS. Furthermore, I propose sRCS that gives more optimized configurations than RCS for individual applications. I demonstrate that sRCS can provide 6% higher geo-mean performance than RCS.

Chapter 6

VR-Scale: Runtime Voltage Regulator Phase Scaling

Most modern commercial processors support two classes of power management mechanisms: power and performance states denoted by C and P states to maximize power efficiency [98]. When a processor core is in one of the C states, it stops operations and enters a sleep or off state to minimize power consumption. The deeper the C state is, the lower power consumption is at the expense of higher performance penalty due to longer latency to exit from the C state. Besides, a processor offers a trade-off between performance and power consumption using various P states. The deeper the P state is, the lower voltage/frequency of the processor core is, consuming lower power at the expense of lower performance. Leveraging diverse C and P states available, the OS applies aggressive runtime power management policies to considerably reduce power consumption of the processor without notably impacting the performance and/or quality of service (QoS); appropriate C and P states are determined based on (i) the performance demand and behavior of executed applications and (ii) the utilization of the processor at a given time interval.

In a platform, one of the most critical components is voltage regulators (VRs). This is because VRs supply necessary power for various platform components such as processors, chipsets, dynamic random access memory (DRAM) modules and storage devices, while they consume a large fraction of total platform power and area. Overall, the VRs consume 22% of total platform power [99] and they

are the second largest platform component next to the DRAM modules, occupying 63% more platform area than the processor, the third largest component [43].

The key design objective of a VR for a processor is to maximize power conversion efficiency (or simply VR efficiency), because the power dissipated by the VR is directly proportional to the power consumed by the processor, the highest power consuming component in platforms, divided by VR efficiency. For example, when supplying 100W for a processor, a VR with 80% efficiency dissipates 25W. At the same time, they must satisfy various operating requirements: delivering stable voltage and large current while supporting fast, accurate, and fine-grained voltage changes for efficient processor power management. To cost-effectively provide such VRs, multi-phase VRs were proposed [100]. A multi-phase VR is comprised of multiple small VRs, each of which operates at a unique phase to share a burden of delivering large current.

In this chapter, I first demonstrate that VR efficiency can significantly vary depending on the amount of current that it delivers to a processor (i.e., load current), output voltage, and the number of VR phases in use. For given load current and output voltage, VR efficiency can vary more than 20% depending on how many VR phases are activated. Second, I exhibit that the load current may significantly change over a long time period but it is also very predictable over a short time period, in particular when a commercial processor runs a parallel application. Third, I show that the processor consumes relatively small current for a notable fraction of runtime for running a parallel application, because various P and C states are aggressively applied by the OS and the processor's power management unit. This in turn leads to very poor VR efficiency for the past platforms where all the VR phases are typically activated unless all the cores in a processor enters low-power states [101, 102]. Finally, I present VR-Scale that dynamically scales the number of active VR phases based on the

predicted load current at runtime. For two Intel® processors executing an emerging parallel application such as PARSEC [56] and DCBench [103], I demonstrate that VR-Scale can reduce the total power consumed by the processor and its VR by 19% and 25%, respectively, without notably impacting the performance.

Some technical briefs from Intel® and HP® indicate that some of their products dynamically scale the number of active VR phases (e.g., [104, 105]). However, neither any further technical detail nor benefit analysis is publically available. To my best knowledge, this is the first study that provides the deep technical insights of processors dynamically scaling the number of active phases, analyzes its benefit using two commercial platforms based on Intel® Ivy Bridge and Haswell processors, and explores its possible implementation. Furthermore, the existing architectural technique only reduce the number of phases when a processor is in idle or low-power states [101, 102], but I also demonstrate a considerable opportunity to reduce the number of phases while an application is actively running. Lastly, my study opens a door for the architecture community to explore dynamic control of other VR knobs from the CPU side, enabling more cost-effective VRs than the VR side does as argued by [106].

Note that many commercial VRs such as the most recent Intel VR specification (i.e., VR12/IMVP7 [102]) support *both circuit- and architecture-level phase scaling techniques* (e.g, [107, 108]). The circuit-level technique is also known as auto phase shedding and a VR *autonomously* changes the number of phases while monitoring its load current. In contrast, the architecture-level technique is directed by the power status indicator (PSI) pins and a VR simply follows a command from the processor and sets the number of phases accordingly. Although these VRs support both techniques, they should be configured to choose either a circuit- or architecture-level technique [107, 108]. Thus, I investigated why the architectural knob to control the number of phases exists regardless of the circuit-

level autonomous phase shedding support, surveying many literatures and interviewing industry experts.

Although we cannot discover any article explaining the reason for supporting both circuit- and architecture-level phase scaling techniques in one VR, but we got some common responses from industry experts and the following is the most detailed response [109]: "The main challenge with autonomous phase shedding (the VR drops/adds phases on its own) is the load attack condition[1]. Multiphase controllers that support autonomous phase shedding feature can have difficulties in supporting very large load attacks with very fast slew rates if multiple phases are turned off when such an event occurs. Without a PSI indicator that preemptively turns all the phases back on before that large load attack we typically get a much larger output voltage droop as a result of that load attack. The more phases are on, the faster the VR can ramp up its output current and the more support it can provide to the power distribution network and prevent output capacitor discharge. There is additional delay associated with the fact that the phases need to be turned back on before ramping up the current. Recent controllers are faster and better, but we still get better regulation performance (i.e., lower voltage droop) when disabling autonomous phase shedding and relying on PSI. Note that in some extreme cases when the phases are not turned back on properly or fast enough we can see over current and catastrophic failure on phase." This suggests that the architecture-level technique can be more cost-effective (i.e., smaller on-chip and/or on-board decoupling capacitors) and more reliable, while the circuit-level technique is useful (at the cost of larger decoupling capacitors) for processors without any advanced phase control mechanism.

[1] A sudden increase of current for a substantial amount of time.

The remainder of this chapter is organized as follows. Section 6.1 discusses the related work. Section 6.2 describes some background on processor power management features, multi-phase VRs, and processor-VR interfaces. Section 6.3 analyzes potential VR efficiency improvement by optimally scaling the number of VR phases for given load current and output voltage. Section 6.4 describes my experimental methodology. Section 6.5 analyzes runtime current consumption of the processor. Section 6.6 describes VR-Scale and analyzes power efficiency improvement for platforms based on two different Intel® processors. Section 6.7 concludes this chapter.

6.1 Circuit- and Architecture-Level Techniques for VR Phase Scaling

Architecture-level techniques: A dynamic phase scaling technique was proposed in a patent [110], where the number of phases is adjusted based on the current P state assuming that the current consumption is proportional to the P state. However, my analysis shows that the processor power and current consumption can be significantly low at even high voltage/frequency P states where all the phases are used. In contrast, my study demonstrates that some phases even at high voltage/frequency P states can be deactivated when the measured power and current consumption is low. Thus, a technique leveraging the power measurement feature of processors is more effective than this invention.

Intel® Xeon® processors and their platforms support an automatic adjustment of the number of active phases in particular when the processors consume low current [104]. HP® ProLiant servers also scale the number of active phases for processors and dual in-line memory modules (DIMMs) based on their power consumption at runtime [105]. While neither further technical details nor benefit analysis is publically available, we could infer from VR phase controller datasheets (e.g., [111]) that the number of active phases is reduced when the voltage identification (VID) value (i.e., the value control the

output voltage of the VR) from the processor is reduced, which is similar to what was proposed in [110]. Furthermore, it was observed that the power consumed by the processor VR is responsible for 10% of the entire platform power consumption when the entire processor (i.e., all the cores in the processor) enters a deep sleep state for a long time period [101]. Hence, the past Intel® Core micro-architecture is implemented to turn off all the phases except one to maximize the VR efficiency for such a case; in other words, all the phases are always activated unless all the cores enter a deep sleep mode. In contrast, my study analyzes the power efficiency improvement of dynamic phase scaling when a processor is in active mode (although some cores enter a sleep model for a short time period). Powell et. al. [112] uses an architectural techniques to reduce high-frequency inductive noise. This technique proposes pipeline muffling to reduce changes in the number of resources being utilized by controlling instruction issue to control di/dt in space.

Circuit-level techniques: Most circuit-level techniques for dynamic phase scaling is focused on controlling the VR circuit (e.g., [112]). This is to minimize any disruption of the VR output voltage (e.g., the voltage supplied to the processor) and satisfy various output voltage requirements when the number of phases is changed.

6.2 Background

6.2.1 Processor Power and Performance States

Modern commercial processors support C and P states to maximize power efficiency [98]. For example, C0, C1, C3, and C6 states denote operating, halt, sleep, and off states. The deeper the C state is, the lower the power consumption is at the expense of higher performance penalty due to longer wake-up latency. The P0 state indicates the maximum sustainable performance (voltage/frequency)

state under the thermal design power (TDP) constraint. Similar to C states, the deeper the P state is, the lower the power consumption is, at the expense of lower performance. Besides, as a part of P states, the processors support turbo (or T) states where cores run faster than their P0 state if they operate below power, current, and temperature specification. To support various P and T states, the VR connected to the processor should be able to quickly vary supply voltage.

6.2.2 Platform Voltage Regulators

One of the most critical issues for VRs is to maximize their power conversion efficiency, which is defined as the ratio of output power to input power. Preferably, a VR should convert the supplied input voltage to the required output voltage without any power loss. However, the VR itself can dissipate a notable amount of power during the voltage conversion steps, delivering less power than it receives.

Generally, a platform includes a power supply unit (PSU) that converts an AC voltage (e.g., 110V) to various DC voltages (e.g., 12V, 5V, 3.3V, 1.8V, etc.) for platform components such as processor, DRAM, chipset, and hard disk drive (HDD). Since the operating voltages of these components are very diverse, the second-level voltage-down converters (or VRs) are also required on a platform as illustrated in Figure 6-1 (left). Such a two-step voltage conversion technique is also commonly used for high efficiency, because directly converting a high AC voltage to low DC voltages is very inefficient. Furthermore, even with on-chip VRs in Intel® processors with Haswell micro-architecture [113], an off-chip VR on the platform is still required, because on-chip VRs share the same manufacturing technology with the processor and thus they cannot directly receive a high DC voltage from the PSU, either [114, 115].

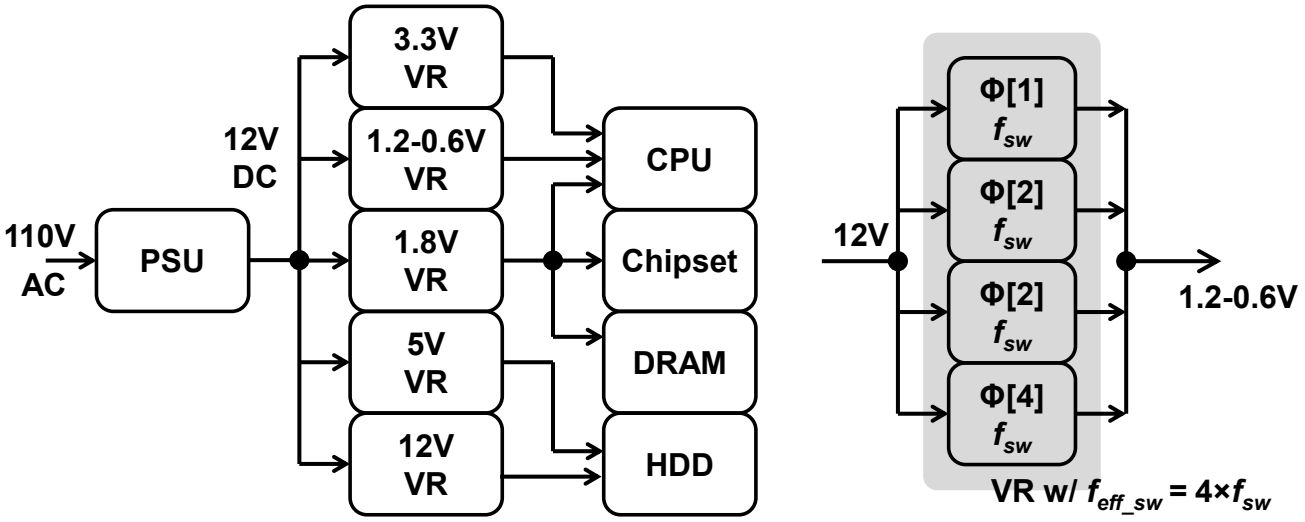


Figure 6-1: (left) Platform power delivery architecture. (right) An example of a 4-phase VR, where each phase (Φ) operates at f_{sw}, with the effective f_{sw} (f_{eff_sw}) = 4×f_{sw}.

VRs are comprised of capacitors, inductors, and transistors as on-/off- switches, and they switch on/off these transistors at certain frequency and duty cycle to regulate the output voltage to a desired level. Such (switching) VRs desire higher switching frequency (denoted by f_{sw}) for lower output voltage fluctuation and faster responses to the changes of load current. However, simply increasing f_{sw} of a single VR circuit leads to higher power loss (i.e., lower efficiency). To provide high efficiency at low switching frequency, the multi-phase switching VR, where multiple basic VR circuits are placed in parallel between the input and load (i.e., processor) as illustrated in Figure 6-1 (right), was proposed [116]. Each of N phases is turned on at equally spaced intervals over 1/ f_{sw}, increasing the effective switching frequency by N times without increasing associated switching losses.

The number of maximum VR phases depends on the maximum load current required by the connected processor and varies from 3 to 8 phases, while typical VRs for *high-performance* processors use as many as 6-8 phases [117, 118, 119]. Moreover, the number of phases and their efficiency is primarily a function of output voltage, load current, and the number of phases used to deliver voltage

and current for the processor (or simply the number of active phases). More parallel phases decrease conduction loss because they decrease VR's effective resistance proportional to the number of parallel phases. However, more parallel phases also increase switching loss because they increase VR's switching capacitance. Therefore, more active phases, which decrease conduction loss, offer higher efficiency for higher load current. In contrast, fewer active phases, which decrease switching loss, provide higher efficiency for lighter load current. To maximize efficiency for varying output voltage and load current, most state-of-the-art multi-phase VRs allow us to adjust the number of active phases [117].

6.2.3 Processor-VR Interface

A typical CPU requires at least 3 or 4 voltage domains (i.e., VRs); cores, I/O devices, and analog components such as phase-locked loops (PLLs) and on-chip sensors need one domain each, and on-chip caches may demand another domain when the voltage domain between the cores and on-chip caches is split [73]. As more components such as GPUs and DSPs are integrated in a single chip [120], more voltage domains and their independent controls are needed.

CPUs often require VRs that can deliver large current while demanding VRs to satisfy various stringent requirements. These requirements include: (i) limited voltage overshoot/undershoot for large transient current and/or large voltage change and (ii) fast and/or fine-grained voltage changes (e.g., 6.25mV/μs) for dynamically changing P states at runtime and/or statically compensating process, voltage, and temperature (PVT) variations at manufacturing time. For the state-of-the-art Intel® processors, these requirements and the interfaces between the processor and the VRs are specified by VR12/IMVP7 pulse width modulation (PWM) specification [102].

To achieve high VR efficiency while cost-effectively satisfying all these stringent requirements, the power control unit (PCU) in a processor needs to closely interact with the connected VRs and control various operating parameters of the VRs at runtime. With a growing number of such power domains per processor, efficient and cost-effective interfaces and communication protocols between a processor and VRs such as PMBus [121] have emerged.

6.3 Efficiency Improvement Opportunity in Multi-Phase VRs

As described in Section 6.2.2, platform VRs for processors use as many as 6-8 phases [117, 118, 119] and their efficiency primarily depends on output voltage, load current, and the number of active phases (denoted by n) used to deliver voltage and current for the processor. To maximize efficiency for varying output voltage and load current, most state-of-the-art multiphase VRs allow us to dynamically adjust the number of active phases [117].

6.3.1 Efficiency versus Load Current and Output Voltage

Figure 6-2 (left) plots the efficiency of a 6-phase VR as a function of load current and the number of active phases at 1.2V output voltage. This VR is optimized to deliver power for an Intel® Ivy Bridge processor that can consume up to 77W at P0 state (i.e., voltage = 1.2V or maximum load current = 77W/1.2V = 65A). For each 3-tuple of load current (incrementing it up to 65A by 1A), voltage (decrementing it from 1.6V to 0.7V by 10mV), and the number of phases (from 1 to 6), I measure the efficiency of a VR and create an *efficiency look-up table* after running SPICE based on a model [54] and the following key VR design parameters that impact efficiency: the switching frequency of each phase (f_{sw} = 300KHz), inductance (L = 360nH) per phase, and parasitic resistance of inductor (r_L = 0.5mΩ) that are derived from a commercial 6-phase VR for an Intel® processor [122]; I validate the

efficiency for the range of output voltage and load current against a commercial switching VR used for Intel® processors [122, 123]. As discussed in Section 6.2.2, more active phases offer higher efficiency for higher load current. In contrast, fewer active phases provide higher efficiency for lighter load current. For example, for 5A load current, using all 6 phases gives only 64% efficiency while using only 1 phase provides 86% efficiency (i.e., 22% higher efficiency than using 6 phases).

Figure 6-2 (right) plots the number of active phases maximizing efficiency as a function of P state (i.e., voltage) and load current, where the maximum load current at each P state decreases as the voltage decreases. To obtain the maximum load current for each P state, I scale the maximum load current value at P0 state based on scaling factors obtained by measuring power consumption of various SPEC benchmarks at each P state (cf. Section 6.4 for the experimental methodology). Figure 6-2

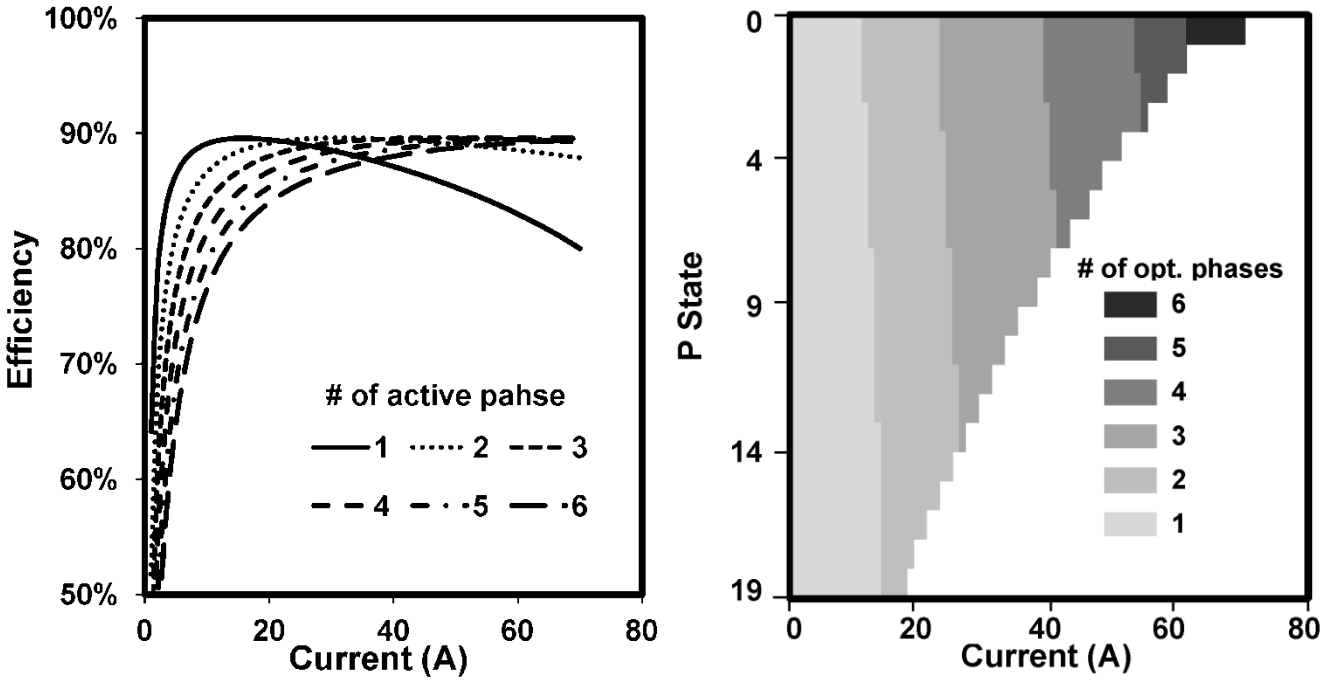


Figure 6-2: Efficiency as a function of current consumed by a processor and *n* at P0 state = 1.2V (left). The optimum number of active phases as a function of a P state and load current of a processor (right).

(right) is obtained by searching the number of phases providing the highest efficiency for each pair of load current and P state in the efficiency look-up table constructed for Figure 6-2 (left). Figure 6-2 (right) clearly shows that the optimum number of active phases, which maximizes the VR efficiency, depends on both output voltage and load current. Thus, when all the phases are always activated (or the number of active phases is poorly chosen), I can observe more than 20% efficiency loss. Thus, it is critical to dynamically scale the number of active phases at runtime, because output voltage and load current of modern processors can significantly vary at runtime.

6.3.2 Impact on Overall Power Efficiency

While most studies consider the power consumption of only the processor, my study takes the power supplied from the PSU to the VR as total power consumption (P_{TOT}). When the processor consumes P_{PROC} (i.e., power delivered for the processor by the VR), P_{TOT} can be represented as follows:

$$P_{TOT}(n) = \frac{P_{PROC}}{\eta(n)} \qquad (6\text{-}1)$$

where n, η, n_{max}, and n_{opt} denote the number of active phases, the VR efficiency (< 1), the maximum number of phases, and the optimum number of active phases, respectively. Then the improvement of total power efficiency is expressed by:

$$1 - \frac{P_{TOT}(n_{opt})}{P_{TOT}(n_{max})} = 1 - \frac{\eta(n_{max})}{\eta(n_{opt})} \qquad (6\text{-}2)$$

Table 6-1: System configuration.

CPU	Intel *i*7-3770K (8MB LLC)	Intel i7-4770 (8MB LLC)
Microarchitecture	Ivy Bridge (4 cores)	Haswell (4 cores)
TDP	77W	88W
Technology	22nm	
Frequency	3.9GHz~1.6GHz	
Memory	DDR3-1600/8GB	

For each pair of P state and load current, I search n_{opt} and apply it to Eq. (2) to compute the improvement. When the load current is less than 5A, optimally scaling n can improve total power efficiency by 25%-63%. The smaller the load current and/or the lower voltage is, the higher the improvement is. I observe that a processor running a single- or multi-threaded application often consumes very small current for a large fraction of runtime. This is because many cores are often idle and thus they often put into very deep P and/or C states; I will demonstrate this in Section 6.5. For such applications, I can see that optimally scaling n can considerably improve total power efficiency.

6.3.3 Impact of Runtime Phase Scaling on Performance and Reliability

Note that modern multi-phase VRs began to allow the processor to choose n [106]. The processor can send a command (i.e., command code 0x04 [106]) and then n to a VR through PMBus [121]. Considering that PMBus operates at 400KHz and use a serial interface such as I2C [124], it takes 60µs to send the command and the parameter to the target VR; each adjustment requires the processor to send an 8-bit (VR) address, an 8-bit (phase change) command, and an 8-bit parameter (i.e., n). After the VR PWM controller (de)activates one or more phases, it takes several f_{sw} cycles (< 30µs) to stabilize the output current; the exact time depends on the L and C values of the VR design.

Table 6-2: P state and corresponding frequency/voltage; the VID values are rounded to the second decimal point.

P state	Freq/Volt	P state	Freq/Volt
Turbo	3.9/1.52	P8	2.7/1.00
	3.8/1/42	P9	2.6/0.97
	3.7/1.32	P10	2.5/0.95
	3.6/1.22	P11	2.4/0.92
P0	3.5/1.20	P12	2.3/0.90
P1	3.4/1.17	P13	2.2/0.87
P2	3.3/1.15	P14	2.1/0.85
P3	3.2/1.12	P15	2.0/0.82
P4	3.1/1.10	P16	1.9/0.80
P5	3.0/1.07	P17	1.8/0.77
P6	2.9/1.05	P18	1.7/0.75
P7	2.8/1.02	P19	1.6/0.72

Thus, a circuit-level autonomous phase scaling technique requires larger on-chip and/or on-board decoupling capacitors to supply the current during the phase change, which increases the cost of the platform but is useful when the processor has no phase control feature. In contrast, an architecture-level phase scaling technique can briefly halt the process during the phase change, requiring smaller decoupling capacitors. When the processor changes n with the voltage, it does not need to halt according to the VR12/IMVP7 specification [102]. However, just changing the number of active phases without changing the voltage requires the processor to halt for 3.3µs [102].

6.4 Experimental Methodology

For evaluations, I use commercial computer systems based on two Intel® processors with SMT enabled. I run all the benchmarks on Linux Ubuntu 12.04 after tuning on the "on-demand" and "menu" governors for the P- and C-state management policies [125]. Table 6-1 tabulates the detailed

configuration of the computer system used for my experiments. I use the performance and energy counters in the processor [126] to measure processor power consumption as well as execution time spent at various P and C states (i.e. P- and C-state residencies) every 1ms. Intel® Ivy Bridge and Haswell processors measure the power consumption using a similar technique demonstrated in [127].

In measuring C-state residency, I consider C0 (i.e., non-sleep state including both active and idle states), C1 (halt), C3 (sleep), and C6 (off) states. In C3 state, the power consumption of each core significantly decreases since the clock generator (i.e., phase-locked-loop (PLL)) of the core is turned off (i.e., no dynamic power) and the voltage is decreased to a bare minimum to retain the memory states (i.e., low leakage power). It takes 60μs for an Intel® Ivy Bridge core to wake-up from C3 state. The wake-up time is mostly spent for re-starting the PLL to provide stable frequency. When a core is in C6 state, the power consumption is almost zero, because the power-gating device for the core is turned off along with the PLL after on-chip caches are flushed. It takes 80μs for an Intel® Ivy Bridge core to wake-up from C6 state. The wake-up time includes turning on the power-gating device, re-starting the PLL, and restoring the architectural state of the core.

I measure the frequency of the processor each 1ms interval by reading a special register in the processor and obtain the corresponding VR output voltage (VID) value by looking up Table 6-2; the measured voltage and power are used to compute the load current (= power/voltage) and the corresponding VR efficiency.

Table 6-3: Mixes of two and four SPEC CPU2006 benchmarks. The benchmarks in gray-colored cells are ones to create mixes of two benchmarks.

Notation	Benchmarks			
mix-00	xalancbmk.r	mcf.r	milc.s	gcc.s
mix-01	sphinx3.a	mcf.r	hmmer.r	soplex.r
mix-02	bzip2.c	zeusmp.z	lbm.l	hmmer.r
mix-03	soplex.r	xalancbmk.r	h264ref.f	astar.r
mix-04	libquantum.r	milc.s	soplex.p	bzip2.c
mix-05	soplex.p	sphinx3.a	soplex.p	bzip2.c
mix-06	zeusmp.z	h264ref.f	gcc.s	soplex.p
mix-07	astar.r	GemsFDTD.r	hmmer.r	gcc.s
mix-08	zeusmp.z	xalancbmk.r	sphinx3.a	soplex.r
mix-09	mcf.r	xalancbmk.r	astar.r	perlbench.d
mix-10	libquantum.r	soplex.p	perlbench.d	hmmer.r
mix-11	soplex.p	milc.s	libquantum.r	zeusmp.z
mix-12	lbm.l	xalancbmk.r	soplex.r	milc.s
mix-13	zeusmp.z	libquantum.r	soplex.r	milc.s
mix-14	gamess.c	perlbench.d	h264ref.f	hmmer.r

I take PARSEC, DCBench, and SPEC CPU2006 benchmark suites [56, 103, 128] for my evaluation. For all the PARSEC benchmarks, I use the native input set with 8 hardware threads. For SPEC CPU2006, I also evaluate 15 mixes of 2 and 4 co-running benchmarks. Table 6-3 summarizes the composition of SPEC CPU2006 benchmarks for each mix where the benchmarks in gray-colored cells are ones to create mixes of two benchmarks. The mixes of four SPEC CPU2006 benchmarks are based on [129]. For each PARSEC or SPEC CPU2006 benchmark, I run it until completion for multiple times. Finally, I repeatedly execute each mix of SPEC CPU2006 benchmarks such that the execution of one benchmark is overlapped with various starting points of the other one or more co-running benchmarks. Table 6-4 tabulates the abbreviation of each PARSEC and DCBench benchmark used in various plots.

Table 6-4: Abbreviation of benchmark names.

PARSEC		DCBench	
Benchmark Name	**Abbreviation**	**Benchmark Name**	**Abbreviation**
blackscholes	black	sort	sort
bodytrack	body	wordcount	word
canneal	cannel	grep	grep
dedup	dedup	naive bayes	bayes
facesim	face	svm	svm
ferret	ferret	k-mean	kmeans
fluidanimate	fluid	fuzzy-k-mean	fkmeans
freqmine	freq	item-based collab. filtering	ibcf
raytrace	ray	frequent pattern growth	fpg
streamcluster	stream	hidden markov model	hmm
swaptions	swap	grep select	grep-s
vips	vips	ranking select	rank-s
x264	x264	user visit aggregation	agreg
		user visit ranking join	rank-j

6.5 Runtime Processor Current Consumption

In this section, I demonstrate that a commercial processor running various benchmarks consumes very low current for a substantial fraction of runtime and analyze the root-cause of consuming low current using various measured statistics.

PARSEC: Figure 6-3 plots the runtime breakdown by the amount of current consumed by an Intel® Ivy Bridge processor running each workload. In each 1*ms* measurement interval, the current consumption is obtained by measuring the power consumption every 1*ms* and dividing it by the voltage associated with the P state of the interval; since a P state changes every 10*ms*, the voltage during each 1*ms* measurement interval is constant. When the processor runs a PARSEC benchmark, it consistently consumes substantially lower current than the maximum current for a large fraction of

runtime. On average, the processor running a PARSEC benchmark consumes less than 5A and 25A (less than 8% and 40% of the maximum current) for almost 60% and 80% of its total runtime, respectively.

Although all four cores are used to execute each parallel benchmark, the power management mechanism aggressively applies deep P and/or C states to maximize power efficiency. For many PARSEC benchmarks using *futex* as a synchronization mechanism, cores spend a considerable fraction of runtime in idle states waiting for synchronizations [130] and they often enter deep C states as shown in Figure 6-4. Similar to Figure 6-3, I measure the residency of each P or C state for each core and obtain the average residency of all four cores every 1*ms*. For example, the cores running *blackscholes*, *bodytrack*, *ferret*, and *vips* spend 60%, 70%, 74%, and 77% of their runtime in C6 state, respectively. Furthermore, they spend 80%, 88%, 62%, and 95% of their runtime in very low voltage/frequency (P16-P20) states, respectively. This leads to relatively low current consumption in most intervals for the PARSEC benchmarks.

DCBench: The processor running a DCBench benchmark consumes somewhat higher current than a PARSEC benchmark, but it still consumes low current for a significant fraction of runtime. I observe

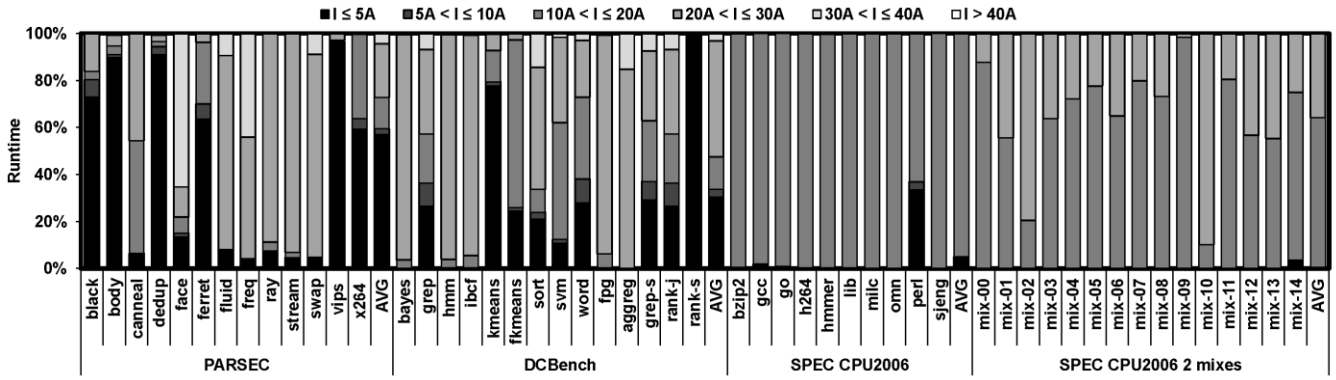


Figure 6-3: Runtime breakdown by the amount of current consumed by the processor.

that the processor running a DCBench benchmark spends less time in C6 state and considerably more time at high voltage/frequency (T0-T3) states. My hypothesis is as follow. Many PARSEC benchmarks experience many synchronization events. Consequently, many cores spend a considerable amount of runtime in idle state. Thus, the power management mechanism often puts these idle cores into deep P and/or C states (e.g., P16-P20 and/or C3-C6). In contrast, many DCBench benchmarks are very data-parallel applications and thus they do not experience many synchronization events. Yet they are often I/O-bound and thus the processor running such a benchmark still spends a notable fraction of their runtime in C6 states and consumes relatively low current. On average, the processor running a DCBench benchmark consumes less than 5A and 25A for almost 30% and 85% of its total runtime, respectively.

SPEC CPU2006: When the processor runs or co-runs one or two SPEC CPU2006 benchmarks at a time, only one or two cores are heavily used while the remaining cores stay in C6 states in most intervals, as depicted by Figure 6-4. Further, the current consumption of the cores is often limited by the maximum operating voltage and/or thermal constraints. Therefore, the current consumption of the processor is relatively low in most intervals. As shown in Figure 6-3, the processor consumes less than 15A (one SPEC CPU2006 benchmark) and 20A (two co-running SPEC CPU2006 benchmarks) for more than 90% of its runtime on average. However, it is expected that the current consumption of the processor co-running four SPEC CPU 2006 benchmarks at a time is fairly high although the measurement is not shown in Figure 6-3. The processor running four SPEC CPU 2006 benchmarks spends a significant fraction of runtime in C0 state while it also spends most of runtime in high voltage/frequency (T0-T3) states if it is in C0 state. On average, the processor consumes less than 25A,

more than 25A but less than 30A, and more than 30A but less than 35A for 25%, 60%, and 16% of their runtime, respectively.

In summary, such low current consumption of a multi-core processor running one multi-threaded benchmark such as PARSEC and DCBench or one or two single-threaded SPEC CPU 2006 benchmarks will lead to very poor conversion efficiency for VRs whose operating parameters are typically set for high current consumption.

6.6 VR-Scale

In this section, I first show how the load current changes over time for various benchmarks. Second, based on this observation, I present VR-Scale, a simple technique that dynamically adjusts the number of active phases considering the load current change over time and evaluate its effectiveness in terms of overall power efficiency improvement and performance impact. Third, I analyze the effectiveness of VR-Scale for improving power efficiency of another commercial processor supporting per-core voltage/frequency scaling using on-chip VRs.

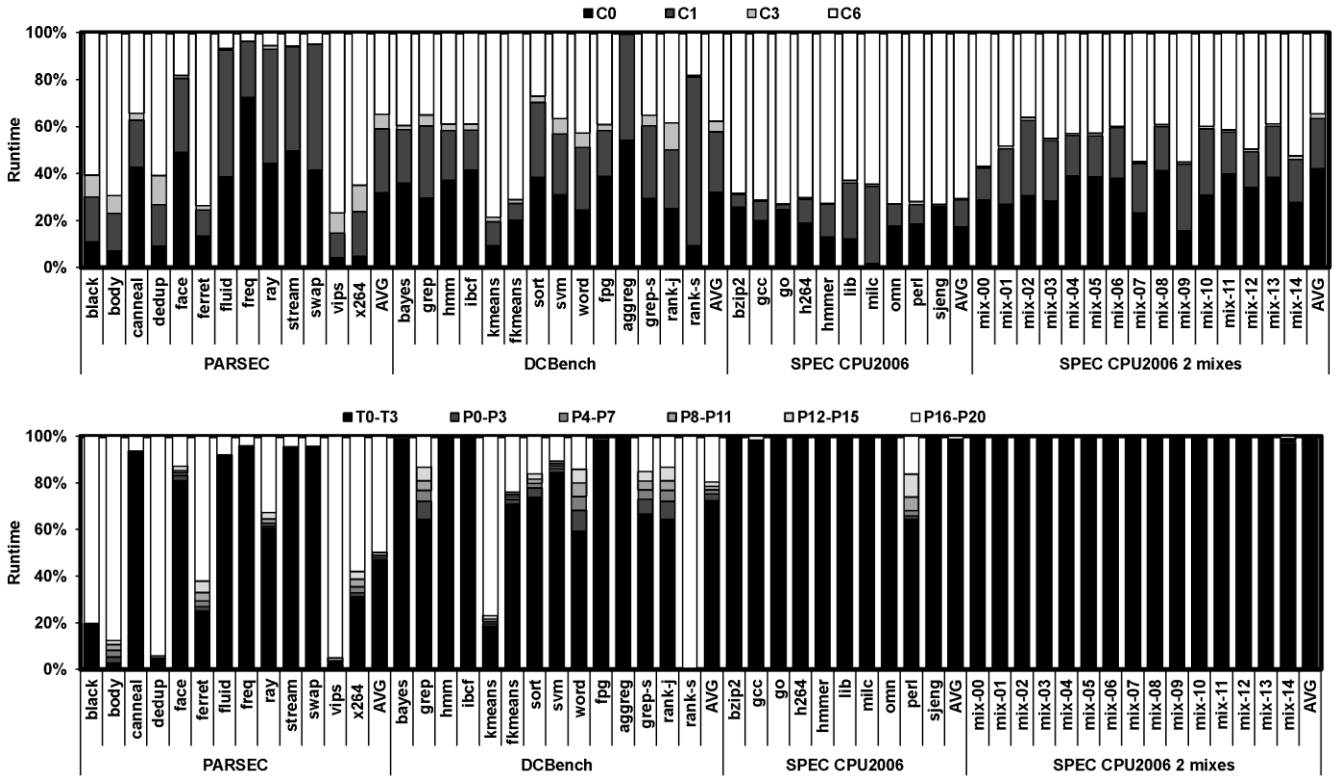


Figure 6-4: Breakdown of processor C and P states (top and botton, respectively).

Finally, I analyze the impact of the limited time resolution of my current measurement (i.e., 1*ms*) using a commercial processor on my power efficiency evaluation because the load current may change significantly and fast while each measured current value is an average value over an 1*ms* period.

6.6.1 Temporal Load Current Change Pattern

One key observation made from this study is that the power consumption of a processor is mainly dominated by the number of cores in C0 state and their P state. This is because fixed power consumption components such as leakage and clocking power dominate the total power consumption of a core in C0 state; I observe that benchmarks showing significant phase changes in performance do

not exhibit as notable phase changes in power consumption in [131] and my experiment using some microbenchmarks.

Figure 6-5 plots the load current, which is measured every 1*ms*, of *bodytrack*, *swaptions*, and *canneal* over time. *bodytrack* exhibits a repetitive load current pattern because it repeats for the same computations for each frame. The load current is very low for a while, because only one core is running while the remaining cores are waiting for synchronizations; it was observed that the serial phase of *bodytrack* dominates the execution time [132]. Then the load current considerably increases for a short time period as the synchronizations are resolved and all the cores resume their execution. *facesim*, *ferret*, *vips*, *x264*, and *raytrace* in PARSEC exhibit a similar temporal load current change pattern. All DCBench benchmarks also show a similar temporal load current change pattern except for *hidden markov model* (*hmm*), *frequent pattern growth* (*fpg*), and *item-based collaborative filtering* (*ibcf*). *canneal* shows a few distinct phases of temporal load current change. *blackscholes*, *dedup*, and *streamcluster* in PARSEC also show a similar temporal load current change pattern. For these benchmarks, the number of running threads, which changes over time, dominantly determines current consumption of the processor. *swaptions* show load current notably fluctuating atop a certain offset over time. The fluctuation is incurred by varying execution phases of a thread while the offset is induced by parallelism that always uses all the cores. Most SPEC CPU2006 benchmarks and their mixes show such a temporal load current change pattern. In fact, SPEC CPU2006 benchmarks show significant phase changes in performance, but they do not exhibit as notable phase changes in power consumption [131] because the fixed power consumption components such as leakage and clocking power dominates the total power consumption of a core in C0 state. This confirms my observation based on the evaluation of SPEC CPU2006 benchmarks.

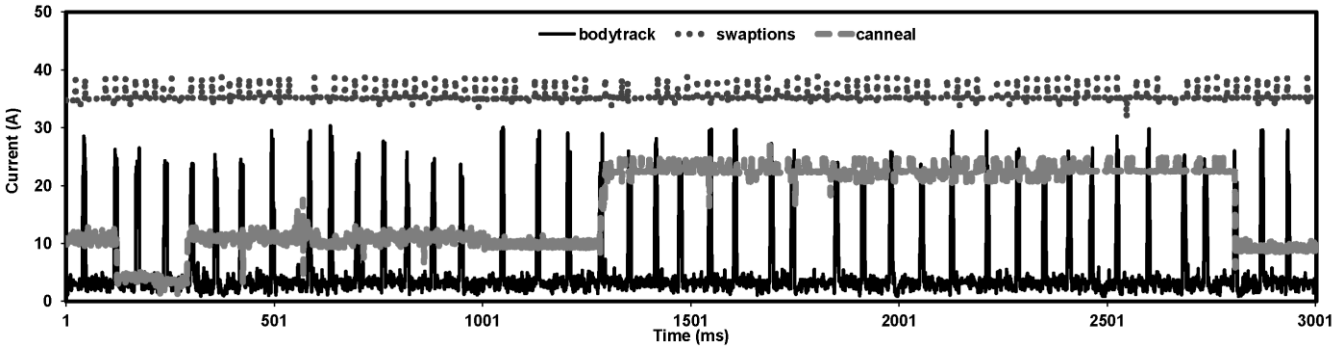


Figure 6-5: Load current change over time. The load current and P-state are measured every 1ms.

Analyzing the temporal load current change patterns of these three classes of benchmarks, I see that the load current in *ms* time granularity is very predictable and stable although it can significantly change over a longer time period (e.g., hundreds *ms*). Furthermore, the VR-Scale's prediction of load current for the next interval does not need to be very accurate since VR-Scale attempts to predict the optimum number of active phases (n_{opt}) where a particular n_{opt} value can offer the highest efficiency for a wide range of load current (e.g., from 12A to 23A for $n_{opt} = 2$), as depicted in Figure 6-2 (right).

6.6.2 Architectural Support and Power Efficiency Evaluation

In Section 6.6.1 I demonstrated that the load current change is not only very stable for a long period but also predictable in each measured period. Thus, I present to use a very simple technique to predict the load current for the next interval. As discussed in Section 6.2.3, the PCU, which also periodically computes the power consumption of the processor at runtime, is responsible to control the number of active phases. Figure 6-6 describes the pseudo code for the PCU to determine n_{opt} based on a look-up table (LUT) approach.

Runtime algorithm: The LUT is a 2-dimensional array indexed by the P-state value and the number of active phases (n); each entry stores the maximum load current value at which a VR can deliver the highest efficiency for a given P-state value and n. Re-examining Figure 6-2 (right), I see that a range of P states often can share the same row in the LUT since the VR efficiency is a far more strong function of load current than voltage.

The number of entries is 16 for the VR used in this study and the LUT entry values can be stored in the platform BIOS chip. Once the LUT is programmed for a given VR, the search algorithm takes at most n_{max} look-ups to determine n_{opt} (i.e., 6 in this study). For example, when a given P-state value is 2, the first row of the LUT is accessed. Each entry in the first row is compared against given load current in ascending order. When the load current value is 25A, the linear search ends at the third step, which implies that n_{opt} is 3. The computational overhead is small enough even for a simple PCU.

```
////////////////////////////////////////////////////////////
// maximum current for a given number of phases to maximize //
// VR efficiency                                            //
////////////////////////////////////////////////////////////
   lut[NUM_PSTATES][NUM_PHASES] = {
   {11, 23, 39, 53, 61, 64},  // p_state = 0-3
   {12, 24, 40, 53},          // p_state = 4-7
   {13, 25, 41, 43},          // p_state = 8-11
   {14, 27}};                 // p_state = 12-16
////////////////////////////////////////////////////////////
// a function to extract the optimum number of active phases  //
////////////////////////////////////////////////////////////
int determine_nopt(int p_state, int current){
     nopt = 0;
     while (lut[p_state/4][n++] <= current);
     return n;
}
```

Figure 6-6: VR-Scale runtime algorithm pseudo code running on PCU to determine n_{opt}.

Accuracy: Figure 6-7 (top) shows how accurately n_{opt} can be determined by two techniques: VR-Scale and the algorithm based on the current P state of the processor [110], compared to the oracle method. In particular, VR-Scale correctly determines n_{opt} for most SPEC CPU2006 benchmarks for nearly 100% of intervals, because the current consumption is very stable throughout execution, as discussed in Section 6.6.1. Although VR-Scale incorrectly determines n_{opt} , the chosen n is very close to n_{opt} (e.g., picking n = 2 for n_{opt} = 1); on average, the difference between n and n_{opt} is less than 0.5 phase for all the benchmarks. In such a case, the negative impact on the power efficiency improvement is very small; the VR efficiency disparity between two neighboring n values is small when the load current slightly exceeds the upper limit for which a particular n value can provide the highest efficiency. In contrast, the algorithm based on the P state often picks n highly deviating from n_{opt} (e.g., picking n = 6 for n_{opt} = 1) since many benchmarks consume low current although running at high voltage/frequency P states (Figure 6-4); on average, the difference between n_{opt} and n at each interval is 2.5, 3.5, and 4.8 phases for PARSEC, DCBench, and SPEC CPU2006, respectively. Consequently, the algorithm based on the P state leads to notably worse power efficiency improvement than VR-Scale for many applications (in particular for running SPEC CPU2006 benchmarks), as demonstrated in the next paragraph.

Power efficiency improvement: Figure 6-7 (bottom) plots the total power efficiency improvement using VR-Scale and the algorithm based on the P state, relative to a naive approach that always uses all 6 phases. I calculate the overall efficiency improvement of each benchmark as follows:

$$1 - \frac{\sum_{i=1}^{N} P_{TOT}(n[i])[i]}{\sum_{i=1}^{N} P_{TOT}(n_{max})[i]} \qquad (6\text{-}3)$$

where $n[i]$ denotes n determined by VR-Scale, the algorithm based on the P state [110], or the oracle algorithm at interval i; $P_{TOT}(n[i])[i]$ (= $P_{PROC}[i]/(\eta(n[i]))$ denotes the sum of processor and VR power consumption (cf. Eq. (6-1) and (6-2) for given $n[i]$ at interval i; $P_{PROC}[i]$ is the processor power consumption measured at interval i; and N is the total number of 1*ms* intervals for a given benchmark. I measure $P_{PROC}[i]$ and voltage every 1*ms* to calculate the corresponding load current at interval i. Then I determine $n[i]$ based on a given algorithm after looking up the efficiency look-up table for the given pair of the load current and voltage. The oracle algorithm can determine n_{opt} at each 1*ms* interval.

On average, VR-Scale can improve the power efficiency (defined by Eq. (3)) by 23% and 14% for running two parallel benchmark suites, PARSEC and DCBench (i.e., average power reduction of 1.4~3.8W and 1.1~3.4W for average processor power consumption of 3~34W and 4~32W), respectively. For running one and two SPEC CPU2006 benchmarks at a time, VR-Scales improves the power efficiency by 12% and 7% (i.e., average power reduction of 1.9~2.9W and 1.6~1.8W for the average processor consumption of 10-17W and 21~25W), respectively. Finally, I showed that the processor co-running four SPEC CPU2006 benchmarks consumes very high current for a considerable fraction of runtime, but VR-Scale can still improve the power efficiency by 4% on average although not being plotted in Figure 6-7.

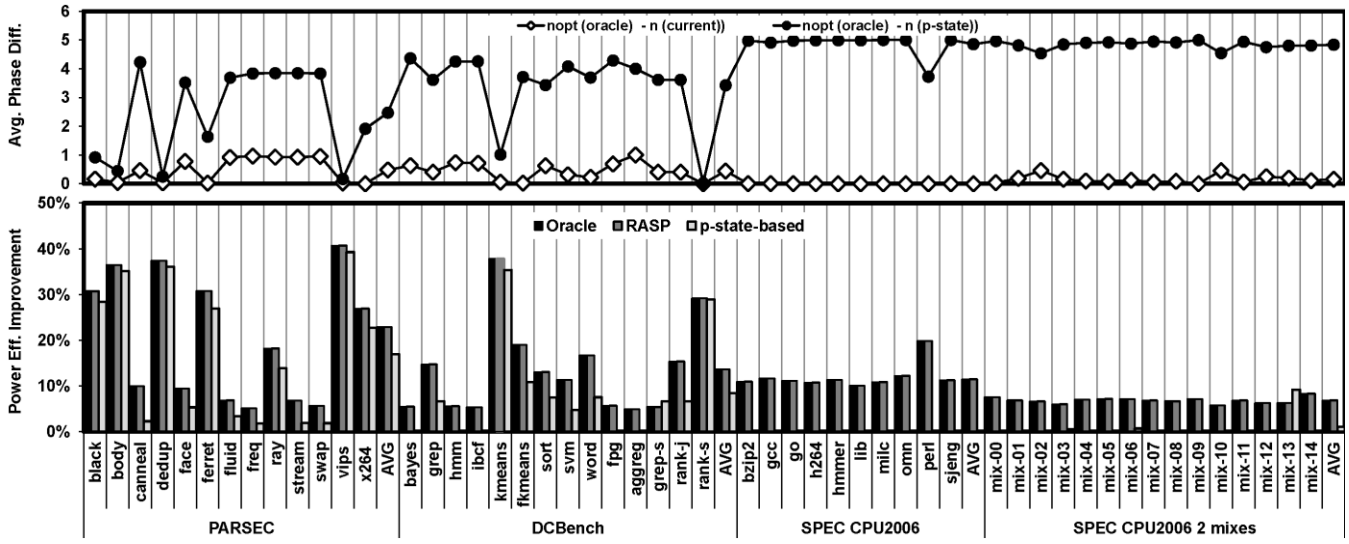


Figure 6-7: Average of n_{opt} by the oracle method - n by current and p-state-based techniques at each interval (top). Power efficiency improvement of current- and p-state-based techniques relative to the VR always activating n_{max} phases (bottom).

On the other hand, the algorithm based on the P state [110] provides notably lower power efficiency improvement than VR-Scale; it improves the power efficiency by 16% and 8%, both of which are about 4% lower than VR-Scale, for running PARSEC and DCBench, respectively. In particular, the algorithm based on the P state practically does not improve power efficiency at all for running one, two, and four SPEC CPU2006 benchmarks at a time while VR-Scale improves power efficiency by 12%, 7%, and 4%, respectively. This is because all SPEC CPU2006 benchmarks mostly run at the highest voltage/frequency P states while they do not consume maximum allowed current.

Performance overhead and reliability: The VR-Scale runtime algorithm is simple and runs on the PCU. Hence, running this algorithm does not incur any CPU performance overhead. Furthermore, while assuming that the processor stalls before changing the number of active phases, I see that the negative performance impact of changing n at runtime is negligible because (i) the time interval is two

orders of magnitude longer than the stall time for significantly changing n at runtime, (ii) such stall events are very infrequent, and (iii) n_{opt} in fact does not change frequently.

It was shown that sudden load-current changes in Figure 6-5 are induced by C-state changes of individual cores and such changes are controlled and anticipated by the PCU. Hence, the PCU can proactively handle such cases by halting the processor until the VR stabilizes before such significant load-current changes (also see Section 6.3.3).

6.6.3 Processor Supported by On-Chip VRs

So far, I have performed my evaluation using an Intel® processor based on Ivy Bridge micro-architecture where a single off-chip VR supplies the power for all four cores. Recently, Intel® introduced a processor based on Haswell micro-architecture that can change each core's voltage/frequency independently using on-chip VRs. Furthermore, Haswell micro-architecture offers more advanced and aggressive power management features than its predecessor such as deeper and more intelligent C states with faster C-state exit latency [114]. Consequently, the average power consumption of the processor based on Haswell micro-architecture is expected to be notably smaller than its predecessor although both processors are manufactured with the same 22nm technology.

In a platform based on Haswell micro-architecutre, however, we still need an off-chip VR between the on-chip VRs and the PSU shown in Figure 6-1; the on-chip VRs share the same manufacturing technology with the processor and thus the on-chip VRs cannot directly accept 12V from the PSU. For example, an off-chip VR receives 12V and supplies 1.8V to the on-chip VRs [114, 115]. Thus, I can apply VR-Scale not only to on-chip VRs but also to the off-chip VR. However, I focus on only the off-chip VR in this study because: (i) the technical details of the on-chip VRs for the Haswell-based

processor such as the number of phases, switching frequency, and inductance are not publically available and (ii) it is likely that the benefit of VR-Scale is limited for on-chip VRs; the number of available phases for each on-chip VR is likely much fewer than the off-chip VR because an on-chip VR connected to a single core needs to deliver much smaller maximum load current than an off-chip VR shared by all the cores.

Figure 6-8 plots the power efficiency improvement of the Ivy Bridge and Haswell processors, as well as the average power consumption ratio between two processors that are manufactured with the same 22nm technology. For this experiment, I use the Haswell processor (i7-4770) that has the same LLC capacity (8MB) and maximum frequency (3.9GHz) as the Ivy Bridge processor (i7-3770K) and similar TDP (84W versus 77W), and the same DDR3-1600 DIMM and storage device. To calculate the VR efficiency for given current and voltage, I scale the measured current consumption with the TDP ratio between two processors (i.e., 77W/88W) because the VR is designed and optimized for 77W TDP. Finally, I re-optimize the off-chip VR such that its efficiency is maximized for fixed 1.8V output voltage when all 6 phases are active for the maximum current.

Due to the aforementioned more advanced and aggressive power management features, the Haswell processor consumes much lower average current than the Ivy Bridge processor. Consequently, VR-Scale for the Haswell processor can improve power efficiency significantly more compared to that for the Ivy Bridge processor. Figure 6-8 shows that VR-Scale for the Haswell processor can improve power efficiency of PARSEC, DCBench, one SPEC CPU2006, and two co-running SPEC CPU2006 benchmarks by 29%, 24%, 22%, and 16%, respectively, which are 6%, 10%, 11%, and 9% higher than VR-Scale for the Ivy Bridge processor, respectively. To explain such higher power efficiency

improvement, I compare the power consumption of two processors. I observe that the average power consumption of the Haswell processor is approximately 40% of the Ivy Bridge processor; which is close to what Intel® reported [114, 133]; the lower the average power is, the lighter the load current is. Thus, VR-Scale for the Haswell processor can offer higher power efficiency than the Ivy Bridge processor.

6.6.4 Impact of Measurement Time Interval

In Section 6.6.2, my evaluation is based on (average) current consumption of a processor over each 1*ms* interval that is the minimum measurement time interval of the Intel® processor. On the other hand, the actual current consumed by the processor may notably vary in each 1*ms* interval. In such a case, my evaluation of power efficiency improvement based on the current consumption values measured every 1*ms* can be less accurate than every e.g., 1μs because VR efficiency depends on the load current. Furthermore, in such a case, n_{opt} determined by VR-Scale for an 1*ms* interval may not be optimal for many sub 1*ms* intervals of the 1*m*s interval. Finally, the 1*ms* time resolution for measurement also

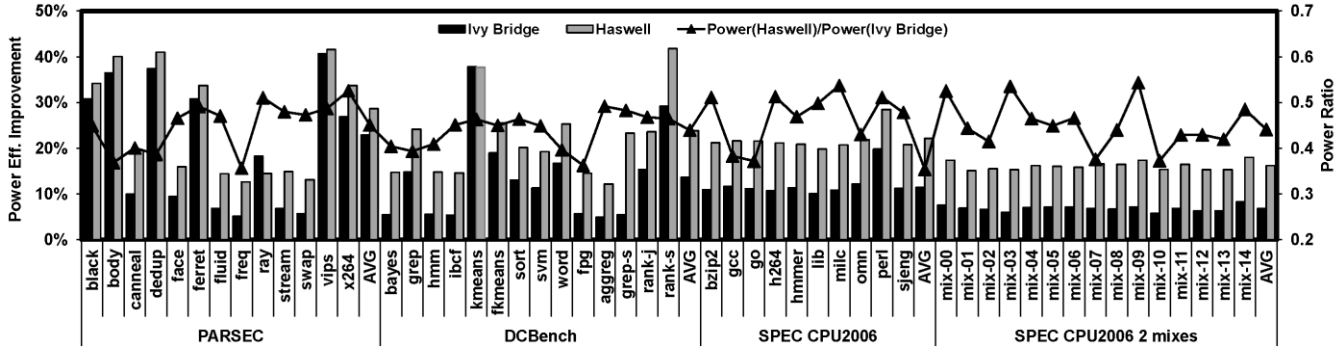


Figure 6-8: Comparison of power efficiency improvement of the Ivy Bridge and Haswell processors, and the average power consumption ratio between two processors. I use 1ms interval for VR-Scale for both microarchitecture.

limits how fast VR-Scale adapts n_{opt} in my evaluation.

To quantify the impact of limited measurement and adjustment time interval, I take a simulation-based sampling approach to capture the effect of faster current change than 1*ms* yet in a reasonable evaluation time. I use gem5 [134] and McPAT [58] that are configured and calibrated to model the Ivy Bridge processor for this evaluation. For each benchmark, I randomly pick 100 1*ms* intervals, estimate an average current value every 10µs using gem5 and McPAT, and compute the power efficiency improvement.

Table 6-5 summarizes the impact of limited current measurement and VR adjustment time intervals on power efficiency improvement. It also shows the degree of load current variations relative to the average current in lieu of σ/µ; each 1*ms* interval is comprised of 100 current values measured every 10µs and I evaluate the standard deviation (σ) and mean (µ) values of 100 load current values for each 1*ms* interval and show the average σ/µ values of 100 1*ms* intervals. For power efficiency evaluations, I can consider three cases: the periods of power efficiency evaluation and n_{opt} adaptation are (i) 1*ms* and 1*ms* (i.e., baseline), (ii) 10µs and 1*ms*, and (iii) 10*µs* and 10*µs*, respectively. (ii) and (iii) are to analyze the impact of limited current measurement and VR adjustment time intervals on evaluation accuracy and further power efficiency improvement. I see that using 1*ms* for measuring the current leads to less than 1% evaluation error compared to using 10µs. Besides, using 10*µs* for measuring the current and adjusting n_{opt} can improve power efficiency by less than 1% compared to using 1*ms*. This is because the current of the processor does not significantly fluctuate relative to the average current value measured over 1*ms* time period, because the fixed power consumption components dominates the total power consumption of a core, as discussed earlier. This argument is also supported by small load current σ/µ values for the benchmarks depicted in Table 6-5; most intervals of SPEC CPU2006

Table 6-5: Impact of limited time resolution of measurement on power efficiency improvement. σ/μ denotes the standard deviation of load current over the average load current of a 1ms interval. “max σ/μ” and “avg σ/μ” are the maximum and average σ/μ values of 100 1ms intervals, respectively.

Benchmarks	Power Efficiency Improvement			σ/μ max/avg
	1ms/1ms	10μs/1ms	10μs/10μs	
bzip2.c	10.53%	10.46%	11.03%	0.14/0.11
gcc.s	11.11%	11.03%	11.80%	0.16/0.11
gobmk	10.69%	10.68%	11.15%	0.15/0.10
h264ref.f	10.17%	10.15%	10.57%	0.14/0.10
hmmer.r	10.34%	10.24%	10.83%	0.13/0.10
libquantum.r	9.68%	9.6%	10.76%	0.13/0.10
milc.s	11.39%	11.36%	11.97%	0.13/0.11
omnetpp	11.57%	11.57%	12.52%	0.14/0.11
perlbench.d	18.95%	17.72%	22.98%	0.24/0.14
sjeng	10.53%	10.51%	10.96%	0.14/0.11
AVG	11.19%	11.17%	12.64%	

benchmarks show at most 10% load current fluctuation relative to the mean load current on average. I also observe a very similar trend for PARSEC benchmarks although I do not show the evaluation in Table 6-5; I could not simulate DCBench benchmarks because many of them require the Ethernet connections to multiple nodes, which cannot be simulated by gem5.

So far I assume that the current change incurred by a processor is instantaneously reflected to the load current change experienced by the VR. However, such a current change by a processor is not immediately and directly translated into a load current change that the VR has to cope with. This is because the resistance (R), inductance (L), and capacitance (C) components of the package and platform power delivery network (PDN) as discussed in [135] and average out the current fluctuations..

Furthermore, I have already discussed that notably power change is primarily incurred by the number of cores in C0 states and the P state of these cores in Section 6.6.1. Note that 1*ms* is the fast time interval that the OS changes the P state [59] due to the overhead of changing the frequency of the PLL is tens of µs [136]. Furthermore, it takes nearly 100µs to enter into and exit from C3 and C6 states. Hence, the current fluctuation is primarily incurred by cores in C0 state and their micro-architectural activity changes, which cannot dramatically change the current consumption of a processor in an 1*ms* interval.

6.7 Chapter Summary

In this chapter, I first demonstrate that: (i) VR efficiency heavily depends on load current (i.e., current consumed by a processor) and a VR operating parameter (e.g., the number of VR phases) at given voltage; (ii) a processor running one or more applications may consume large current for some periods but mostly consumes small current due to aggressive power management; and (iii) unless all the cores in a processor are in sleep states, all VR phases are activated, leading to poor VR efficiency for small load current. Second, I present VR-Scale, a low-cost architecture-level technique that dynamically scales the number of active phases based on the predicted load current for the next interval. VR-Scale only requires to run a simple runtime algorithm using the PCU in commercial processors every 1ms.

My evaluations using two commercial platforms based on Intel® Ivy Bridge and Haswell processors show that VR-Scale reduces the total power consumption of a processor and its VR by 19% and 25%, respectively, with negligible performance impact for two classes of parallel applications. Finally, I show that VR-Scale can offer high power efficiency improvement than the algorithm that

determines the optimum number of active phases based on the current P state of the processor; many applications run at the highest voltage/frequency P state while consuming low current because not all the cores are always running. Finally, my study opens a door for the architecture community to explore dynamic controls of other VR knobs such as VR switching frequency and adaptive voltage positioning from the CPU side, enabling more cost-effective VRs than the VR side does as argued by [106].

Chapter 7

Summary and Future Work

In this chapter, I first summarize my main contributions in this dissertation, and then discuss some potential future research directions.

LDO (chapter 3): In this dissertation, I demonstrated a cost-effective power delivery technique to support per-core voltage domains for power-constrained processors. Most commercial processors only have one chip-wide voltage domain for dynamic voltage and frequency scaling (DVFS), because splitting the voltage domain into per-core voltage domains and powering them with multiple off-chip voltage regulators (VRs) incurs a high cost for the platform and package designs. Although using on-chip switching VRs can be an alternative solution, integrating high-quality inductors and cores on the same chip has been a critical technical challenge.

I proposed a very low-cost VR regulator that exploits existing on-chip per-core power-gating (PCPG) devices available in most commercial processors. The proposed technique considerably reduces the cost of VRs because VR shares its most expensive component with the PCPG device, while providing power efficiency as high as the state-of-art onchip switching VRs. Furthermore, I proposed a DVFS algorithm that is optimized for the proposed VRs. The proposed DVFS algorithm considers the limited voltage difference between output voltage values (VO) and the input voltage values (VI) of the VRs to support high efficient power delivery. In order to minimize the potential negative impact of the limited voltage range of the VRs in the proposal, the DVFS algorithm leverages

(i) within-die core-to-core process variations that can lead to different frequency and power consumption trade-offs between cores even at the same operating voltages (ii) thread migration at runtime to satisfy the performance requirement of the running threads. Using these techniques, I showed that the power efficiency of LDO is comparable to switching voltage regulator at much lower area cost.

Low-Voltage On-Chip Cache Architecture using Heterogeneous Cell Sizes (chapter 4): In this dissertation, I explored architectural techniques to provide low minimum operating voltage cost-effectively. Although DVFS has been one of the most successful power-reduction techniques, it is limited to some minimum operating voltage (i.e., V_{DDMIN}). Thus, reducing the V_{DDMIN} will help to have larger voltage scaling window for DVFS which can improve the power efficiency. The large scale memory structure in processors such as last-level cache (LLC) often determines the V_{DDMIN} of the processor due to impact of process variation on the reliability of static random access memory (SRAM) at low voltages. Larger SRAM cells, that are less sensitive to process variability, allow the use of lower V_{DDMIN}, but at high area costs.

To provide low V_{DDMIN} with low area overheads, I proposed an LLC architecture comprised of heterogeneous cell sizes designed for high-performance multicore processors. This exploits (i) the DVFS characteristics of workloads running on high-performance processors, (ii) the trade-off between SRAM cell size and V_{DDMIN}, and (iii) the fact that latency between off-chip memory and on-chip core at lower voltage/frequency operating states is reduced. I exploited these characteristics to deliver both high-performance and low V_{DDMIN} by architecting an LLC consisting of a spectrum of cell sizes. For low-power operation the proposed LLC exclusively uses large cells that exhibit low failure rates at low

voltages, and therefore the LLC can operate at low VDDMIN. On the other hand, for high-performance operation it operates at a high enough voltage that the failure rate of even small cells of the LLC is sufficiently low for their use, providing the needed LLC capacity. As operating voltage decreases, subsets (e.g., ways in a set-associative LLC) of cells are disabled in order of size, starting with the smallest. Since at lower voltages a processor must run at lower frequencies, the frequency gap between on-chip cores and off-chip memory decreases. The result is that the performance penalty of having a smaller LLC at low voltage/frequency states is much smaller than it would be at high voltage/frequency states.

Unlike my proposed LLC, a conventional architecture needs to use large cells across the entire LLC to provide the same low VDDMIN as the proposed architecture. This will either require larger die area for the LLC capacity or result in smaller the LLC capacity for the same die area. My proposed LLC architectures provide the same maximum performance and VDDMIN as the conventiona architecture, while reducing the total LLC cell area by 15%-19% with negligible average runtime increase.

DRCS (Chapter 5): With aggressive technology scaling, the nominal voltage of transistors has decreased while process variability has considerably increased. Consequently, the range of voltage scaling will vanish significantly diminishing the power reduction benefit of DVFS. Faced with such a challenge, the third contribution of my research pursues an architectural power efficiency mechanism as a DVFS alternative that trades performance with power consumption through jointly scaling the resources of a core and the number of operating cores. In addition, I developed a runtime policy that

can effectively manage and leverage the proposed architectural mechanism to maximize performance under a power constraint.

VR-Scale (chapter 6): A voltage regulator (VR), which is the main component of power delivery sub-system, plays a critical role in efficiently delivering stable voltage and large current for a processor, as well as supporting efficient power management. A VR dissipates high power, which is directly proportional to the power consumed by the processor. Today's VRs provide multiple knobs that can be adjusted to increase the VR efficiency for different processor specifications.

In this study, I demonstrated that VR efficiency heavily depends on load current (i.e., current delivered to a processor) and a VR operating parameter (e.g., the number of active phases) at a given voltage. I also found that a processor running a parallel application mostly consumes small current due to aggressive power management. However, when the processor is in an active state, all the VR components are always activated in the platform which leads to poor VR efficiency during most runtime. Therefore, I presented VR-Scale that dynamically scales the number of active phases based on the predicted load current for the next interval. My evaluations based on two Intel® processors running emerging parallel applications showed that VR-Scale can reduce the total power consumed by both a processor and its VR with negligible performance impact.

7.1 Directions for Future Work

7.1.1 Hierarchical LDO VR

As future processors will have more cores (i.e., many-core processors), they will potentially exhibit higher core-to-core IPC variations by running mixes of single- and multi-threaded applications. This

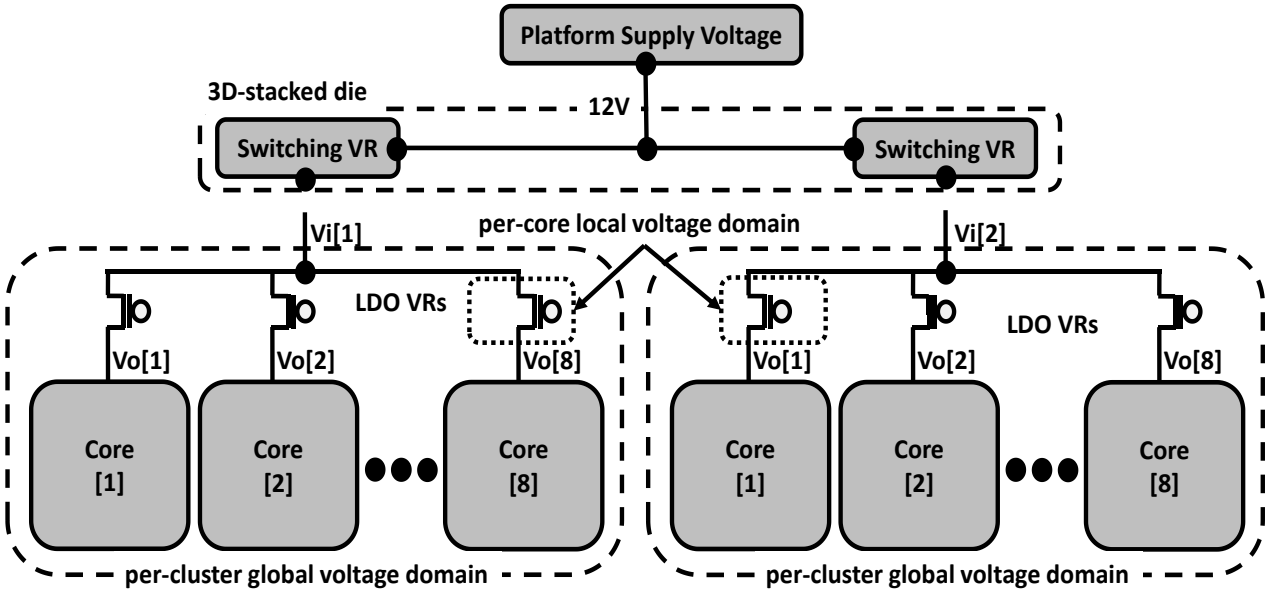


Figure 7-1: Hierarchical VR scheme.

will result in wider $V_I - V_O$ range and thus worse power efficiency ($MIPS^3/W$) when using LDO VRs, due to their poor power conversion efficiency. On the other hand, in many-core processors, it will not be practical to provide a large number of switching VRs as integrating high-quality on-chip inductor becomes more challenging with technology scaling while integrating on-package inductors will not be a scalable solution for a larger number of cores due to the physical constraint. In such a case, we can use a hiearchical VR scheme where a switching VR provides a shared voltage domain for a subset (or a cluster) of cores, and LDO VRs provide per-core voltage domains within each cluster, as illustrated in Figure 7-1. This hiearchical power delivery architecture allows us to support cost-effective per-core voltage domains, which can minimize both the number of switching VRs while maximizing the power efficiency of individual LDO VRs.

7.1.2 Memory Hierarchy at Low VDD

At low voltages where the trade-offs are different we need to rethink about processor components in particular memory hierarchy. For example, leakage power becomes dominant portion of total power at the low-voltage operating conditions. In addition, at lower voltages/frequencies, the latency gap between memory and cores decreases. In these conditions, larger caches or traditional multiple layers of cache hierarchy might not be beneficial any more. On the other hand, at high-performance mode, there is a demand for larger caches. Therefore, to achive both high-performance and low-power, one potential solution is to use a heterogeneous cell size cache hiearchy in the format of non-uniform cache architecture (NUCA). In this design, we use larger cache cells for the banks near the processor and smaller cache cells for those far from cores.

7.1.3 Dynamic Resource and Core Scaling Expansion

In many-core systems, the oppurtunity to execute multiple applications will increase. Running multiple independent applications can exhibit more substantial core-to-core performance variations compared to multi-threaded applications. Consequently, applying uniform resource and core scaling will not be a suitable solution. To expand the idea of resource and core scaling, one potential solution is to partition cores into multiple groups, each running one (multi-threaded) application. Each group then can borrow power from the other groups using resource scaling. Besides, a processor with more cores provide an oppurtunity to explore a wider range of processor configurations to satisfy performance requirements.

7.1.4 VR-Scale Expansion

Today's VRs provide multiple knobs that can be adjusted to increase the VR efficiency for different processor specifications. These knobs can change the efficiency of VR. My research presented a circuit/architecture solution to control one of these knobs (i.e., number of active phases). My research opens a door for the architecture community to explore dynamic controls of other VR knobs such as VR switching frequency and adaptive voltage positioning from the CPU side, enabling more cost-effective VRs than the state-of-the-art VR.

References

[1] H. Ghasemi, S. Draper and N. Kim, "Low-voltage on-chip cache architecture using heterogeneous cell sizes for high-performance processors," in *IEEE/ACM Int. Symp. on High-Peformance Computer Architecture (HPCA)*, 2011.

[2] H. Ghasemi, A. Sinkar, M. Schulte and N. Kim, "Cost-effective Power Delivery to Support Per-core Voltage Domains for Power-constrained Processors," in *IEEE Design Automation Conf. (DAC)*, 2012.

[3] H. R. Ghasemi and N. S. Kim, "RCS: runtime resource and core scaling for power-constrained multi-core processors," in *International conference on Parallel architectures and compilation*, Edmonton, 2014.

[4] W. Kim, M. Gupta, G.-Y. Wei and D. Brooks, "System level analysis of fast, per-core DVFS using on-chip switching regulators," in *IEEE/ACM Int. Symp. on High-Perf. Comp. Arch. (HPCA)*, 2008.

[5] R. Teodorescu and J. Torrellas, "Variation-Aware Application Scheduling and Power Management for Chip Multiprocessors," in *IEEE/ACM Int. Symp. on Comp. Arch. (ISCA)*, 2008.

[6] J. Li and J. Martinez, "Dynamic power-performance adaptation of parallel computation on chip multiprocessors," in *IEEE/ACM Int. Symp. on High-Perf. Comp. Arch. (HPCA)*, 2006.

[7] W. Kim, M. Gupta, G.-Y. Wei and D. Brooks, "System level analysis of fast, per-core DVFS using on-chip switching regulators," in *IEEE/ACM Int. Symp. on High-Perf. Comp. Arch. (HPCA)*, 2008.

[8] S. Eyerman and L. Eeckhout, "Fine-Grained DVFS Using On-Chip Regulators," *ACM Trans. Archit. Code Optim. (TACO),* vol. 8, no. 1, pp. 1-24, Feb 2011.

[9] S. Herbert and D. Marculescu, "Variation-aware dynamic voltage/frequency scaling," in *IEEE/ACM Int. Symp. on High Performance Comp. Arch. (HPCA)*, 2009.

[10] S. Dighe, S. Vangal, P. Aseron, S. Kumar, T. Jacob, K. Bowman, J. Howard, J. Tschanz, V. Erraguntla, N. Borkar, V. De and S. Borkar, "Within-Die Variation-Aware Dynamic-Voltage-Frequency-Scaling With Optimal Core Allocation and Thread Hopping for the 80-Core TeraFLOPS Processor," *IEEE J. of Solid-State Circuits (JSSC),* vol. 46, no. 1, pp. 184-193, Jan 2011.

[11] K. Rangan, G.-Y. Wei and D. Brooks, "Thread motion: fine-grained power management for multi-core systems.," in *IEEE/ACM Int. Symp. on Comp. Arch. (ISCA)*, 2009.

[12] K. Zhang, "A 3-GHz 70-Mb SRAM in 65-nm CMOS technology with integrated column-based dynamic power supply," *IEEE J. of Solid-State Circuits,* vol. 41, no. 1, pp. 146-151, 2006.

[13] K. M and e. al., "PVT-variations and supply-noise tolerant 45nm dense cache arrays with diffusion-notch free (DNF) 6T SRAM cells and dynamic multi-Vcc circuits," in *In Proc. of IEEE VLSI Circuit Symp.*, 2008.

[14] S. Schuster, "Multiple word/bit line redundancy for semiconductor memories," in *IEEE J. of Solid-State Circuits*, 1978.

[15] N. Verma and A. Chandrakasan, "A 65nm 8T sub-Vt SRAM employing sense-amplifier redundancy," in *IEEE Int. Solid-State Circuit Conf.*, 2007.

[16] R. Joshi, "6.6+ GHz low Vmin, read and half select disturb-free 1.2 Mb SRAM," in *IEEE VLSI Circuit Symp.*, 2007.

[17] L. Chang, "A 5.3GHz 8T-SRAM with operation down to 0.41V in 65nm CMOS," in *IEEE VLSI Circuit Symp*, 2007.

[18] I. Chang, J. Kim and K. Roy, "A 32kb 10T subthreshold SRAM array with bit-interleaving and differential read scheme in 90nm CMOS," in *IEEE Int. Solid-State Circuit Conf*, 2008.

[19] J. Kulkarni, "A 160mV robust Schmitt trigger based subthreshold SRAM.," *IEEE J. of Solid-State Circuits,* vol. 42, no. 10, pp. 2303-2313, 2007.

[20] T. May and M. Woods, "Alpha-particle-induced soft errors in dynamic memories," in *IEEE T. on Electron Devices*, 1979.

[21] D. Roberts and e. al., On-chip cache device scaling limits and effective fault repair techniques in future nanoscale technology, In Proc. of IEEE Euro micro Conf. on Digital System Design, Architecture, Method, and Tools, 2007.

[22] C. Wilkerson and e. al., Trading off cache capacity for reliability to enable low-voltage operation, In Proc. of IEEE Int. Symp. on Computer Architecture, 2008.

[23] J. Abella and e. al., "Low Vccmin fault-tolerant cache with highly predictable performance," in *In Proc. of IEEE Int. Symp. on Microarchitecture*, 2009.

[24] Z. Chishti and e. al., "Improving cache lifetime reliability at ultra-low voltage," in *In Proc. of IEEE Int. Symp. on Microarchitecture*, 2009.

[25] R. Dreslinski and e. al., "Reconfigurable energy-efficient near threshold cache architecture," in *In Proc. of IEEE Int. Symp. on Microarchitecture*, 2008.

[26] A. Iyer and D. Marculescu, "Microarchitecture-level power management," *IEEE T. Very Large Scale Integration System (TVLSI),* vol. 10, no. 3, pp. 230-239, Jun 2003.

[27] D. Albonesi, R. Balasubramonian, S. Dropsho, S. Dwarkadas, E. Friedman, M. Huang, V. Kursun, G. Magklis, M. Scott, G. Semeraro and P. Bose, "Dynamically Tuning Processor Resources with Adaptive Processing," *IEEE Computer,* vol. 36, no. 12, pp. 49-58, Dec 2003.

[28] B. Lee and D. Brooks, "Efficiency trends and limits from comprehensive microarchitectural adaptivity," in *ACM Int. Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2008.

[29] V. Kontorinis, A. Shayan, D. Tullsen and R. Kumar, "Reducing peak power with a table-driven adaptive processor core," in *IEEE/ACM Int. Symp. on Microarchitecture (MICRO)*, 2009.

[30] H. Homayoun, V. Kontorinis, A. Shayan, T.-W. Lin and D. Tullsen, "Dynamically heterogeneous cores through 3D resource pooling," in *IEEE/ACM Int. Symp. on High-Peformance Computer Architecture (HPCA)*, 2012.

[31] Y. Watanabe, J. Davis and D. Wood, "WiDGET: Wisconsin decoupled grid execution tiles," in *IEEE/ACM Int. Symp. on Computer Architecture (ISCA)*, 2010.

[32] D. Gibson and W. David, "Forwardflow: A Scalable Core for Power-Constrained CMPs," in *International Symposium on Computer Architecture*, Saint-Malo, 2010.

[33] K. Khubaib, M. Suleman, M. Hashemi, C. Wilkerson and Y. Patt, "MorphCore: An Energy-Efficient Microarchitecture forHigh Performance ILP and High Throughput TLP," in *IEEE/ACM Int. Symp. on Microarchitecture (MICRO)*, 2012.

[34] P. Petrica, A. Izraelevitz, D. Albonesi and C. Shoemaker, "Flicker: A Dynamically Adaptive Architecture for Power Limited Multicore Systems," in *IEEE/ACM Int. Symp. Computer Architecture (ISCA)*, 2013.

[35] M. Suleman, M. Qureshi and Y. Patt, "Feedback-driven threading: power-efficient and high-performance execution of multi-threaded workloads on CMPs," in *ACM Int. Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2008.

[36] R. Moore and B. Childers, "Using utility prediction models to dynamically choose program thread counts," in *IEEE Int. Symp. Performance Analysis of Systems and Software (ISPASS)*, 2012.

[37] R. Bitirgen, E. Ipek and J. d Martinez, "Coordinated Management of Multiple Interacting Resources in Chip Multiprocessors: A Machine Learning Approach," in *IEEE/ACM Int. Symp. on Microarchitecture (MICRO)*, 2008.

[38] K. Meng, R. Joseph, R. Dick and L. Shang, "Multi-optimization Power Management for Chip Multiprocessors," in *ACM Int. Conf. on Parallel Architectures and Compilation Techniques (PACT)*, 2008.

[39] K. Ma, X. Li, M. Chen and X. Wang, "Scalable Power Control for Many-core Architectures Running Multi-threaded Applications," in *IEEE?ACM Int. Symp. on Computer Architecture (ISCA)*, 2011.

[40] D. Brooks, P. Bose, S. Schuster, H. Jacobson, P. Kudva, A. Buyuktosunoglu, J. Wellman, V. Zyuban, M. Gupta and P. Cook, "Power-aware microarchitecture: design and modeling challenges for next-generation microprocessors," *IEEE Micro,* vol. 8, no. 6, pp. 26-44, Nov/Dec 2000.

[41] Intel Corporation, "Intel® Turbo Boost Technology 2.0," [Online]. Available: http://www.intel.com/technology/turboboost/index.htm.

[42] S. Sarangi, B. Greskamp, R. Teodorescu, J. Nakano, A. Tiwari and J. Torrellas, "VARIUS: A Model of Process Variation and Resulting Timing Errors for Microarchitects," *IEEE T. on Semiconductor Manufacturing,* vol. 21, no. 1, pp. 3-13, Feb 2008.

[43] Intel Corporation, "Intel Workstation Board S975XBX2 Technical Product Specification," 2006.

[44] P. Hazucha, G. Schrom, J. Hahn, B. Bloechel, P. Hack, G. Dermer, S. Narendra, D. Gardner, T. Karnik, V. De and S. Borkar, "A 233-MHz 80%-87% efficient four-phase DC-DC converter utilizing air-core inductors on package," *IEEE J. of Solid-State Circuits (JSSC),* vol. 40, no. 4, pp. 838-845, Apr 2005.

[45] L. E. Mosley, "Power Delivery Challenges for Multi-Core Microprocessors," in *Capacitor and Resistor Technology Symp. USA (CARTSUSA)*, 2008.

[46] P. Hazucha, S. Moon, S. G., F. Paillet, D. Gardner, S. Rajapandian and T. Karnik, "High Voltage Tolerant Linear Regulator with Fast Digital Control for Biasing Integrated DC-DC Converters," *IEEE J. of Solid-State Circuits,* vol. 42, no. 1, pp. 66-73, Jan 2007.

[47] S. Rusu, S. Tam, H. Muljono, J. Stinson, D. Ayers and C. J., "A 45 nm 8-Core Enterprise Xeon® Processor," *IEEE J. of Solid-State Circuits (JSSC),* vol. 45, no. 1, pp. 7-14, Jan 2010.

[48] N. Kim, J. Seomun, A. L. J. Sinkar, T. Han, K. Choi and Y. Shin, "Frequency and yield optimization using power gates in power-constrained designs.," in *IEEE/ACM Int. Symp. on Low Power Electronics and Design (ISLPED)*, 2009.

[49] Y. Hoskote, S. Vangal, A. Singh, N. Borkar and S. Borkar, "A 5-GHz Mesh Interconnect for a Teraflops Processor," *IEEE Micro,* vol. 27, no. 5, pp. 51-61, Sep/Oct 2007.

[50] W. Fu and A. Fayed, "A feasibility study of high-frequency buck regulators in nanometer CMOS technologies," in *IEEE Dallas Circuits and Systems Workshop (DCAS)*, 2009.

[51] P. Hazucha, T. Karnik, B. Bloechel, C. Parsons, D. Finan and S. Borkar, "Area-Efficient Linear Regulator With Ultra-Fast Load Regulation," *IEEE J. of Solid State Circuits (JSSC),* vol. 40, no. 4, pp. 933-940, Apr 2005.

[52] J. Klein, 2006. [Online]. Available: http://www.fairchildsemi.com/an/AN/AN-6005.pdf.

[53] J. Gjanci and M. H. Chowdhury, "A Hybrid Scheme for On-Chip Voltage Regulation in System-On-a-Chip (SOC)," *IEEE T. on Very Large Scale Integration (VLSI) Systems,* no. 99, pp. 1 - 11, Oct 2010.

[54] J. Lee, G. Hatcher, L. Vandenberghe and C. K. Yang, "Evaluation of Fully-Integrated Switching Regulators for CMOS Process Technologies," *IEEE T. on Very Large Scale Integration (VLSI) Systems,,* vol. 15, no. 9, pp. 1017-1027, Sep 2007.

[55] M. Martin, D. Sorin, B. Beckmann, M. Marty, M. Xu, A. Alameldeen, K. Moore, M. Hill and D. Wood, "Multifacet's general execution-driven multiprocessor simulator (GEMS) toolset," *SIGARCH Comput. Archit. News,* vol. 33, no. 4, pp. 92-99, Nov 2005.

[56] Princeton University, [Online]. Available: http://parsec.cs.princeton.edu/.

[57] K. Aygun, M. Hill, K. Eilert, K. Radhakrishnan and A. Levin, "Power delivery for high-performance microprocessor," *Intel Technology J.,* vol. 9, no. 4, pp. 273-283, Nov 2005.

[58] [Online]. Available: http://www.hpl.hp.com/research/mcpat.

[59] Microsoft, [Online]. Available: http://msdn.microsoft.com/en-us/windows/hardware/gg463252.aspx.

[60] M. Clinton, "Variability and SRAM design," IEEE Int. Solid-State Circuit Conf. Microprocessor Forum, 2008.

[61] http://www.spec.org/power_ssj2008.

[62] A. Garg and P. Dubey, " Fuse area reduction based on quantitative yield analysis and effective chip cost," in *In Proc. of IEEE Int. Symp. on Defect and Fault Tolerance in VLSI Sys*, 2006.

[63] W. J and e. al., "The asynchronous 24-MB on-chip level-3 cache for a dual-core Itanium® family processor," in *In Proc. of IEEE Int. Solid-State Circuit Conf*, 2006.

[64] A. J and e. al., "Low Vccmin fault-tolerant cache with highly predictable performance," in *In Proc. of IEEE Int. Symp. on Microarchitecture*, 2009.

[65] C. Z and e. al., "Improving cache lifetime reliability at ultra-low voltage," in *In Proc. of IEEE Int. Symp. on Microarchitecture*, 2009.

[66] A. Alameldeen, C. Mauer, M. Xu, P. Harper, M. Martin, D. Sorin, M. Hill and W. D.A., "Evaluating Non-Deterministic Multi-Threaded Commercial Workloads," in *Comp. Arch. Evaluation using Commercial Workloads (CAECW)*, 2002.

[67] C. J and e. al., "Physical modeling and prediction of the matching properties of MOSFETs," in *In Proc. of IEEE European Solid-State Device Research Conf.*, 2004.

[68] Z. K and e. al., "A 3-GHz 70-Mb SRAM in 65-nm CMOS technology with integrated column-based dynamic power supply," in

IEEE J. of Solid-State Circuits, 2006.

[69] S. Zhou and e. al., " Minimizing total area of low-voltage SRAM arrays through joint optimization of cell size, redundancy, and ECC," in *In Proc. of IEEE Int. Symp. on Computer Design*, 2010.

[70] S. Rusu, H. Muljono and B. Cherkauer, "Itanium 2 Processor 6M: Higher Frequency and Larger L3 Cache," in *IEEE Micro*, 2004.

[71] "http://www.eas.asu.edu/~ptm".

[72] J. Wuu and e. al., "The asynchronous 24-MB on-chip level-3 cache for a dual-core Itanium® family processor," in *In Proc. of IEEE Int. Solid-State Circuit Conf*, 2006.

[73] M. Khellah, N. Kim, J. Howard, G. Ruhl, Y. Ye, J. Tschanz, D. Somasekhar and N. Borkar, "A 4.2Ghz, 130Mb/cm2, Dual-Vcc SRAM in 65nm CMOS Featuring Active Power Management with Autonomous Compensation of PVT Variation & Aging Impacts," in *IEEE Int. Symp. on Solid Circuit Conf. (ISSCC)*, 2006.

[74] Naveh, A., "Power and thermal management in the Intel Core Duo Processor," *Intel Technology J.,* vol. 10, no. 2, pp. 109-122, 2006.

[75] "http://www.microsoft.com/whdc/system/pnppwr/powermgmt/".

[76] M. Weiser and e. al., "Scheduling for reduced CPU energy," in *In Proc. of USENIX Conf. on Operating Systems Design and Implementation*, 1994.

[77] S. Zhuravlev, S. Blagodurov and A. Fedorova, "Addressing shared resource contention in multicore processors via scheduling," in *In Proc. of ACM Architectural Support for Programming Languages and Operating Systems*, 2010.

[78] S. Herbert and e. al., "Analysis of dynamic voltage/frequency scaling in chip-multiprocessors," in *In Proc. IEEE Intl. Symp. on Low Power Electronics and Design*, 2007.

[79] "http://www.hpl.hp.com/research/cacti".

[80] "http://download.micron.com/downloads/misc/ddr3_power_calc.xls".

[81] S. Li, J. Ahn, R. Strong, J. Brockman, D. Tullsen and N. Jouppi, "McPAT: an integrated power, area, and timing modeling framework for multicore and manycore architectures," in *IEEE/ACM Int. Symp. on Microarchitecture (MICRO)*, 2009.

[82] A. Alameldeen, C. Mauer, M. Xu, P. Harper, M. Martin, D. Sorin, M. Hill and W. D.A., "Evaluating Non-Deterministic Multi-Threaded Commercial Workloads," in *IEEE Computer Architecture Evaluation Using Commercial Workloads Workshop (CAECW)*, 2002.

[83] Standard Performance Evaluation Corporation, [Online]. Available: http://www.spec.org/omp/.

[84] C. Bienia and K. Li, "PARSEC 2.0: A New Benchmark Suite for Chip-Multiprocessors," in *Workshop on Modeling, Benchmarking and Simulation (MoBS)*, 2009.

[85] N. Barrow-Williams, C. Fensch and S. Moore, "A communication characterisation of SPLASH-2 and PARSEC," in *IEEE Int. Symp. on Workload Characterization (IISWC)*, 2009.

[86] N. Fredrickson, A. Afsahi and Y. Qian, "Performance characteristics of openMP constructs, and application benchmarks on a large symmetric multiprocessor," in *ACM Int. Conf. on Supercomputing (ICS)*, 2003.

[87] J. Tschanz, S. Narendra, Y. Ye, B. Bloechel, S. Borkar and V. De, "Dynamic sleep transistor and body bias for active leakage power control of microprocessors," *IEEE J. of Solid-State Circuits (JSSC),* vol. 38, no. 11, pp. 1838-1845, Nov 2003.

[88] D. Foley, M. Steinman, B. Branover, G. Smaus, A. Asaro, S. Punyamurtula and L. Bajic, "AMD'S Llano Fusion APU," in *IEEE HOTCHIPS*, 2011.

[89] S. Damaraju, V. George, S. Jahagirdar, T. Khondker, R. Milstrey, S. Sarkar, S. Siers, I. Stolero and A. Subbiah, "A 22nm IA multi-CPU and GPU System-on-Chip," in *IEEE ISSCC*, 2012.

[90] [Online]. Available: http://ark.intel.com/products/27287/Intel-Xeon-Processor-7140M-16M-Cache-3_40-GHz-800-MHz-FSB.

[91] 2011. [Online]. Available: http://ark.intel.com/products/63696.

[92] I. Park, C. L. Ooi and T. N. Vijaykumar, "Reducing Design Complexity of the Load/Store Queue," in *IEEE/ACM Int. Symp. on Microarchitecture (MICRO)*, 2003.

[93] C. Isci, A. Buyuktosunoglu and M. Martonosi, "Long-term workload phases: duration predictions and applications to DVFS," *IEEE Micro,* vol. 25, no. 5, pp. 39-51, Sep-Oct 2005.

[94] E. Perelman, G. Hamerly, M. Van Biesbrouck, T. Sherwood and B. Calder, "Using SimPoint for accurate and efficient simulation," in *ACM SIGMETRICS*, 2003.

[95] J. Lau, S. Schoenmackers and B. Calder, "Transition phase classification and prediction," in *IEEE/ACM Int. Symp. on High-Peformance Computer Architecture (HPCA)*, 2005.

[96] R. Strong, J. Mudigonda, J. Mogul, N. Binkert and D. Tullsen, "Fast switching of threads between cores," *ACM Oper. Syst. Rev.,* vol. 43, no. 2, pp. 35-45, Apr 2009.

[97] D.-C. Juan, Y.-T. Chen, M. Lee and S.-C. Chang, "An Efficient Wake-Up Strategy Considering Spurious Glitches Phenomenon for Power Gating Designs," *IEEE T. Very Large Scale Integration System (TVLSI),* vol. 18, no. 2, pp. 246 -255, Feb 2010.

[98] [Online]. Available: http://www.acpi.info/.

[99] S. Pawlowski, "Driving Towards Cloud 2015: A technology Vision to Meet the Demands of Cloud COmputing Tomorrow," 2011.

[100] X. Zhou, Z. X., J. Liu, P.-L. Wong, J. Chen, H.-P. Wu, L. Amoroso, F. Lee and D. Chen, "Investigation of candidate VRM topologies for future microprocessors," in *IEEE Applied Power Electronics Conf. and Expo. (APEC)*, 1998.

[101] J. Allarey, V. George and S. Jahagirdar, "Power Management Enhancements in the 45nm Intels," *Intel Technology Journal,* vol. 12, no. 3, pp. 169-178, Oct 2008.

[102] Intel Corporation, *VR12/IMVP7 Pulse Width Modulation (PWM) Specification,* 2009.

[103] Z. Jia, L. Wang, J. Z. L. Zhan and L. C., "Characterizing Data Analysis Workloads in Data Centers," in *IEEE Int. Symp. on Workload Characterization (IISWC)*, 2013.

[104] Intel Corporation, [Online]. Available: http://www.intel.com/content/www/us/en/servers/technologies/efficient-power.html.

[105] HP, *Power efficiency and power management in HP ProLiant servers.*

[106] D. Freeman, "Digital Power Control Improves Multiphase Performance," *Power Electronics Technology,* pp. 2-3, Dec 2007.

[107] Intersil, "ISL6367 Green Hybrid Digital Dual 6+1 Phase PWM Controller for VR12/IMVP7 Applications With SMBus/PMBus/I2C and AUTO Phase," [Online]. Available: http://www.intersil.com/en/products/power-management/computing-power-vrm-imvp/multiphase-controllers/ISL6367.html.

[108] ST, "PM6764, PM6766 VR12.5 digital multiphase controller with PMBus," [Online]. Available: http://www.stmicroelectronics.com.cn/st-web-ui/static/active/cn/resource/technical/document/data_brief/DM00110477.pdf.

[109] J. Gentillet, *Personal communication,* Oracle.

[110] J. Jenne, "Dynamic CPU Voltage Regulator Phase Shedding". U.S.A. Patent US20110320838 A1, Dec 2011.

[111] "NCP5392Q 2/3/4--Phase Controller for CPU Applications".

[112] P. Zumel, C. Fernnndez, A. de Castro and O. Garcia, "Efficiency improvement in multiphase converter by changing dynamically the number of phases," in *Power Electronics Specialists Conference*, 2006.

[113] Intel, "New Microarchitecture for 4th Gen Intel Core Processor Platforms".

[114] Intel, "Technology Insight: Intel Haswell Platform," 2014.

[115] ASUS, 2014. [Online]. Available: http://rog.asus.com/244672013/labels/featured/introduction-to-fully-integrated-voltage-regulators-fivr-on-maximus-vi.

[116] X. Zhou, P. Xu and F. Lee, "A Novel Current-sharing Control Technique for Low-voltage High-current Voltage Regulator Module Applications," *IEEE T. on Power Electronics,* vol. 15, no. 6, pp. 1153-1162, Nov 2000.

[117] ON Semiconductor, "Programmable Multi-Phase Synchronous Buck Converter," 2008. [Online]. Available: http://www.onsemi.com/pub_link/Collateral/ADP4100-D.PDF.

[118] W. Qiu, C. Cheung, S. Xiao and G. Miller, "Power Loss Analyses for Dynamic Phase Number Control in Multiphase Voltage Regulators," in *IEEE Applied Power Electronics Conference and Exposition (APEC)*, 2009.

[119] [Online]. Available: http://www.techpowerup.com/reviews/MSI/Z97_GAMING_5/.

[120] A. Branover, D. Foley and M. Steinman, "AMD Fusion APU:Llano," *IEEE Micro,* vol. 32, no. 2, pp. 28-37, Mar-Apr 2012.

[121] [Online]. Available: https://pmbus.org.

[122] International Rectifier, [Online]. Available: http://www.irf.com/technical-info/whitepaper/pswus03vrmdesign.pdf.

[123] Texas Instrument, [Online]. Available: http://www.ti.com/product/TPS51631/technicaldocuments.

[124] [Online]. Available: www.i2c-bus.org.

[125] "CPU frequency and voltage scaling code in the Linux(TM) kernel," [Online]. Available: https://www.kernel.org/doc/Documentation/cpu-freq/governors.txt.

[126] Intel, Intel® 64 and IA-32 Architectures Software Developer's Manual., 2011.

[127] W. Huang, C. Lefurgy, W. Kuk, A. Buyuktosunoglu, M. Floyd, K. Rajamani, M. Allen-Ware and B. Brock, "Accurate Fine-Grained Processor Power Proxies," in *IEEE/ACM Int. Symp. on Microarchitecture (MICRO)*, 2012.

[128] Standard Performance Evaluation Corporation, "SPEC CPU2006," 2006.

[129] A. Jaleel, W. Hasenplaugh, M. Qureshi, J. Sebot, J. S. Steely and J. Emer, "Adaptive Insertion Policies for Managing Shared Caches," in *ACM Int. Conf. on Parallel Architectures and Compilation Techniques (PACT)*, 2008.

[130] J. Demme and S. Sethumadhavan, "Rapid Identification of Architectural Bottlenecks via Precise Event Counting," in *IEEE/ACM Int. Symp. on Computer Architecture (ISCA)*, 2011.

[131] V. Spiliopoulos, A. Sembrant and S. Kaxiras, "Power-Sleuth: A Tool for Investigating your Program's Power Behavior," in *Int. Symp. on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOT)*, 2012.

[132] O. Itzhak, I. Keidar, A. Kolodny and U. Weiser, "Performance scalability and dynamic behavior of Parsec," in *Workshop on Systems for Future Multicore Architectures*, 2014.

[133] S. Anthony, May 2013. [Online]. Available: http://www.extremetech.com/computing/156739-intel-haswell-will-draw-50-less-power-than-ivy-bridge.

[134] N. Binkert, B. Beckmann, G. Black, S. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell and M. Shoaib, "The gem5 simulator," *SIGARCH Comput. Archit. News,* vol. 39, no. 2, pp. 1-7, May 2011.

[135] J. Leng, T. Hetherington, A. El-Tantawy, S. Gilani, N. Kim, T. Aamodt and V. Reddi, "GPUWattch: Enabling Energy Optimizations in GPGPUs," in *IEEE/ACM Int. Symp. on Computer Architecture (ISCA)*, 2013.

[136] A. Bashir, L. J., K. Ivatury, N. Khan, N. Gala, N. Familia and Z. Mohammed, "Fast lock scheme for phase-locked loops," in *Custom Integrated Circuits Conf.*, 2009.

[137] S. Gilani, N. Kim and M. Schulte, "Scratchpad memory optimizations for digital signal processing applications.," in *Design Automation and Test in Europe (DATE)*, Grenoble, 2011.

[138] K. Aygun, M. Hill, K. Eilert, K. Radhakrishnan and A. Levin, "Power delivery for high-performance microprocessor," *Intel Technology J.,* vol. 9, no. 4, pp. 273-283, Nov 2005.

[139] S. Sarangi, B. Greskamp, R. Teodorescu, J. Nakano, A. Tiwari and J. Torrellas, "VARIUS: A Model of Process Variation and Resulting Timing Errors for Microarchitects," *IEEE T. on Semiconductor Manufacturing,* vol. 21, no. 1, pp. 3-13, Feb 2008.

[140] C. Isci, A. Buyuktosunoglu, C.-Y. Cher, P. Bose and M. Martonosi, "An Analysis of Efficient Multi-Core Global Power Management Policies: Maximizing Performance for a Given Power Budget," in *IEEE/ACM Int. Symp. on Microarchitecture (MICRO)*, 2006.

[141] J. Lee, V. Sathisha, M. Schulte, K. Compton and N. Kim, "Improving Throughput of Power-Constrained GPUs Using Dynamic Voltage/Frequency and Core Scaling," in *ACM Int. Conf. on Parallel Architectures and Compilation Techniques (PACT)*, 2011.

[142] J. Li and J. Martinez, "Dynamic power-performance adaptation of parallel computation on chip multiprocessors," in *IEEE/ACM Int. Symp. on High-Peformance Computer Architecture (HPCA)*, 2006.

[143] T. Lanier. [Online]. Available: http://www.arm.com/files/pdf/at-exploring_the_design_of_the_cortex-a15.pdf.

[144] J. Martinez and E. Ipek, "Dynamic Multicore Resource Management: A Machine Learning Approach," *IEEE Micro,* vol. 29, no. 5, pp. 8-17, Sep/Oc 2009.

[145] R. Cochran, C. Hankendi, A. Coskun and S. Reda, "Pack & Cap: Adaptive DVFS and Thread Packing Under Power Caps," in *IEEE/ACM Int. Symp. on Microarchitecture (MICRO)*, 2011.

[146] X. Wang, K. Ma and Y. Wang, "Adaptive Power Control with Online Model Estimation for Chip Multiprocessors," *IEEE T. on Parallel and Distributed Systems (TPDS),* vol. 22, no. 10, pp. 1681-1696, Oct 2011.