Techniques for Multicore Power and Thermal Management

James Donald

A Dissertation
Presented to the Faculty
of Princeton University
in Candidacy for the Degree
of Doctor of Philosophy

Recommended for Acceptance
by the Program in
Department of Electrical Engineering

June 2007

Abstract

Power dissipation is now a primary limiting factor in the design of future microprocessors. As a consequence, effective control of on-chip temperature is continually becoming more costly. Most transistor failure mechanisms increase exponentially with temperature. While cooling mechanisms such as fans can modulate temperature, such solutions can be expensive, noisy, and have further reliability issues. Furthermore, due to non-uniform power density, non-uniform temperature profiles may create localized heating in the form of hotspots. In future process technologies, the power problem is exacerbated with relative increases in leakage power as opposed to dynamic power, and significant variation in leakage power.

This thesis explores policies for power and thermal management in modern processors. Such techniques can increase overall reliability while mitigating the cost of other cooling solutions. In particular, I approach power management with parameter variation from an analytical view, develop novel dynamic thermal management techniques for multithreaded processors, and explore a fairly exhaustive design space for thermal management in multicore processors. The overall contributions of this work are concepts, policies, analyses, and frameworks for management of power, temperature, variation, and performance on multicore processor platforms.

This thesis can be divided into two major parts. The first proposes a power management scheme for parallel applications. Rather than assuming predictable power characteristics, this study takes into account the natural variation in deep-submicron technologies. It is shown that while power variation can offset power/performance predictability, several benchmarks have convenient parallelism properties that allow a flexible range of variation of as much as $\pm 98\%$ of the target core design power. The second part of this thesis covers thermal management techniques for several processor design scenarios. I first propose an adaptive thermal management policy for single-core processors based on simultaneous multithreading. This adaptive technique

based on controlling priorities between two threads is shown to increase performance for a thermally constrained core by on average 30%. I then further extend this to the multicore processor paradigm, whereby spatial locality allows for robust policies managing multiple hotspots across several cores. My final policy combines process migration with DVFS for a 2.6X performance improvement.

Acknowledgements

Throughout my years in this program I have received much assistance from many whom I would like to offer my thanks. These people have helped me through revising paper drafts, technical assistance, administrative work, and in many intangible ways.

First, I would like to thank all my past and current group members, including Abhinav Agrawal, Abhishek Bhattacharjee, David Brooks, Eric Chi, Gilberto Contreras, Gila Engel, Sibren Isaacman, Canturk Isci, Maria Kazandjieva, Manos Koukoumidis, Vincent Lenders, Ting Liu, Hide Oki, Matt Plough, Chris Sadler, Yong Wang, Carole Wu, Qiang Wu, Fen Xie, and Pei Zhang. Working alongside my labmates has truly had an impact on this work, and I hope that my presence has also made an impression on theirs as well.

With my above labmates I engaged in countless hours of technical discussions to further my knowledge and development. On top of this though, I had to seek outside experts throughout my various projects. Some of these helpful researchers include Murali Annavaram, Woongki Baek, Ken Barr, Chris Bienia, Jayaram Bobba, Jonathan Chang, Kaiyu Chen, Jeff Gonion, Steven Johnson, Ben Lee, Jian Li, Yingmin Li, Eugene Otto, Sanjay Patel, David Penry, Ram Rangan, Vipin Sachdeva, Bob Safranek, Kevin Skadron, Nathan Slingerland, and Neil Vachharajani. Furthermore, a good number of such skilled researchers from other groups were also available to me in person in our own office including Xuning Chen, Noel Eisley, Shougata Ghosh, Kevin Ko, Bin Li, Vassos Soteriou, and Ilias Tagkopoulos. Through such friends I not only received valuable technical assistance, but also engaged in insightful and enjoyable conversations throughout our long days of work.

I would further like to thank my good friends Su Kim, Jennifer Oh, and Bernard Wu. Their support and encouragement outside of the office has always helped me to keep going.

I am indebted to the electrical engineering departmental staff well as the engi-

neering quad staff who have always been on duty to assist me in times of need. I greatly appreciate the hours or days of administrative work done for my sake by Sarah Braude, Linda Dreher, Anna Gerwel, Sarah Griffin, Jennifer Havens, Jo Kelly, Tamara Thatcher, Stacey Weber, Karen Williams, and Meredith Weaver. In addition, the technical staff for engineering as well as the Office of Information Technology have been a tremendous help in overcoming many technical obstacles. For this I thank John Bittner, Gene Conover, Kevin Graham, Curt Hillegas, Dennis McRitchie, Jay Plett, and Rita Saltz.

I would also like to thank the various entities that have funded my graduate work in part over the years. This includes NSF, Intel, SRC, and GSRC for their direct funding, as well as SIGARCH, IEEE TCCA, SIGDA for their helpful membership benefits, publications, and travel grants.

I would especially like to thank my dissertation committee including David August, George Cai, Sharad Malik, and Kai Li for their assistance with this thesis. Last but not least, I would like thank a key member of my committee, my own advisor Margaret Martonosi. Her patience and guidance that has carried me through these challenging years in graduate school.

Contents

Chapter 1

Introduction

1.1 Background

In the past decade, power consumption has become the foremost challenge in the advancement of the microprocessor industry. Until now, Moore's Law has observed a doubling of the number of transistors per chip approximately every 1.5 years. Along with this exponential increase in transistor count comes an exponential increase in power density [6, 29]. Figure 1.1 depicts this progression among several major desktop processors according to Intel's roadmaps [65].

Power and energy consumption can hinder electronics in a number of ways. In the mobile sector, increased energy usage reduces battery life. Recently in the server markets, the direct energy price has become a significant cost factor for datacenters. Another issue resulting from power consumption is that of on-chip temperature management. With the continual increases in power density, cooling costs rise accordingly. Although cooling solutions such as fans or liquid cooling are the prime mechanism for temperature management, more advanced cooling mechanisms represent further costs as well as additional points of failure. In 1998, it was estimated that beyond a certain threshold the cost of cooling for microprocessors increases by $1 for each

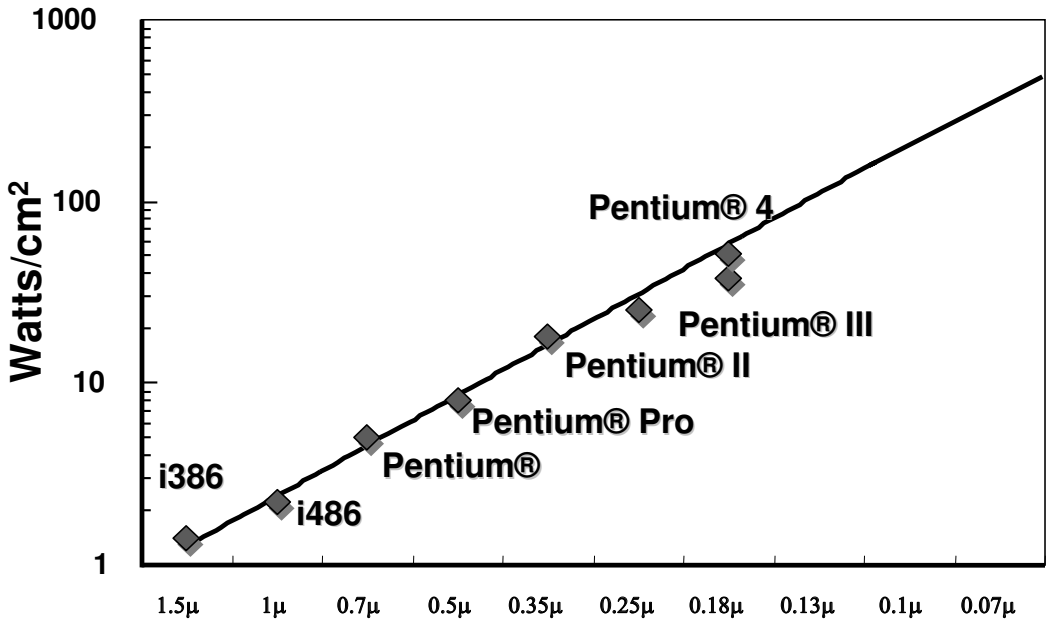


Figure 1.1: Power density of Intel microprocessors across several generations. The x-axis shows the progression of shrinking process technologies.

additional watt [78].

This thesis focuses largely on temperature management techniques. In other words, not only do I present techniques for reducing total power, but I also explore the temperature-aware design space. Temperature-aware design involves taking into account the spatial and transient characteristics of thermal effects. This raises a number of issues not entirely visible when looking only at total power consumption or power density.

1.2 Motivation and Problem Statement

To motivate the challenges in temperature-aware design, here I present measurements taken from real hardware to demonstrate the range of thermal characteristics across different benchmarks selected from the SPEC 2000 suite [33]. These were taken at room temperature on a notebook utilizing a Pentium M Banias 1.5 GHz processor

Figure 1.2: Dell notebook running Linux used for temperature measurements.

and running Red Hat Linux 7.3 with its kernel upgraded to version 2.6.11. Using the Advanced Configuration and Power Interface (ACPI) I read the temperature off a single thermal diode at the edge of the processor [1]. A picture of this setup is shown in Figure 1.2.

I first compiled all benchmarks with base settings using gcc version 2.96 for C programs and Intel Fortran Compiler version 9.0 for Fortran programs. Before running any benchmark the computer was allowed to sit idle briefly and confirm that it had reached its steady-state idle temperature. Once each benchmark run was launched, after one minute I polled the processor temperature repeatedly. Most programs reach a relatively-stable steady-state temperature, and these per-program, steady-state temperatures are shown in Table 1.1 (a). Not all benchmarks gravitate towards a single-steady temperature, however, and Table 1.1 (b) lists the programs where temperatures continually rise and fall throughout execution. As shown, processor steady-state temperatures for such benchmarks can differ by as much as 12°.

(a) Temperatures of stable benchmarks.

benchmark	category	steady-state temperature (º C)
gzip	SPECint	70
mcf	SPECint	59
parser	SPECint	67
twolf	SPECint	67
mesa	SPECfp	65
swim	SPECfp	62
lucas	SPECfp	63
sixtrack	SPECfp	71

(b) Temperature ranges for benchmarks without a steady temperature.

benchmark	category	temperature range (º C)
bzip2	SPECint	67-72
ammp	SPECfp	58-64
facerec	SPECfp	65-71
fma3d	SPECfp	61-67

Table 1.1: Measured processor temperatures on a Pentium M Banias notebook.

These real-system findings are consistent with simulation work by other sources. For example, `gzip` and `bzip2` are two of the hottest integer benchmarks [18] and `sixtrack` is one of the hottest floating point benchmarks [32, 66]. Also, `mcf` is by far the coolest due to its memory-bound execution. Both its overall IPC and temperature are relatively kept low when a limited L2 cache is provided [51], as in this case where the Banias processor provides only 1 MB.

Overall, these real-system measurements show first that applications have quite distinct thermal profiles, and second, that the time-varying nature of applications and workloads warrants truly dynamic approaches to thermal management such as those I explore in Chapters 3 and 4. While this infrasructure cannot physically measure the spatial thermal variations within a core, it can be surmised that CMPs running

multiprogrammed combinations of these applications will show spatial variations at the core-to-core level at least, and likely within the core as well. In fact, such spatial variation of temperature has been strongly confirmed by infra-red emission microscopy (IREM) measurements in industry [26].

The importance of power and thermal management has spawned much research on power reduction through modified process technologies, design methodologies, packing technologies, cooling solutions, and architectural techniques. This thesis particularly examines microarchitectural techniques for power and thermal management. Furthermore, in this area I focus on multicore processors, representing an even more challenging design space than tested in the above physical measurements. In addition to multicore processors, Chapter 3 focuses on multithreaded processors which utilize simultaneous multithreading (SMT) [82]. In recent years, both multicore and multithreaded processors have become ubiquitous, thus representing a the fundamental architectural paradigm for future microprocessors.

The following section describes prior work in the fields of power and thermal management for multicore and multithreaded processors, while Section 1.4 provides an overview of this thesis and its contributions.

1.3 Related Work

This section gives an overview of recent research in the fields of power and thermal management. The intent is to give a summary of prior techniques and show how the ideas presented in this thesis build upon and complete several of these areas. In each of the later chapters, I also provide more detailed overviews of related work specific to each chapter.

Even before multicore processors came into production, there was much prior research on architectural techniques for reducing power in single-core processors.

Wattch [10] and PowerTimer [8] are frameworks for using architectural simulators for power estimation. Using Wattch, Brooks et al. examine several fundamental dynamic thermal management techniques including global clock gating and dynamic frequency scaling [9]. More advanced techniques for uniprocessor power management include formal management of multiple-clock domains [86] and dynamic compilation for frequency and voltage control [88]. One shortcoming of these past works however, is that they have been targeted at only uniprocessors running single-threaded applications. Currently, however, the single-threaded single-processor paradigm no longer represents the dominant processor architecture in most market sectors.

In this decade alone, this industry has seen the introduction of processors exhibiting SMT, chip multiprocessing (CMP), as well as processors formed from a hybrid of the two. Much research on power management has adapted accordingly, leading to studies of the power and energy efficiencies of SMT and CMP. Kaxiras et al. compare SMT to CMP for mobile phone workloads [42], while Sasanka et al. perform a similar comparison for multimedia workloads [67]. Li et al. explored various design tradeoffs for performance and power, as well as temperature, for various SMT and CMP designs [51]. Most of these prior works, as well as the latter thrust of this thesis, focus only on workloads consisting of single-threaded programs. Chapter 2 contributes to this area by exploring parallel applications. Furthermore, although there have recently been a few works on power management for parallel programs [4, 23, 48, 49], the approach taken by my work in Chapter 2 significantly differs by exploring the concept of process variation.

Going beyond power and performance tradeoffs as explored in many earlier works, the second part of this thesis examines such issues from the perspective of thermal management. As shown earlier, the concept of temperature introduces factors of locality not entirely critical when optimizing merely for total power or energy. Furthermore, the ideas in this thesis seek to take advantage of the natural partitioning

and flexibility provided in SMT and CMP architectures. One work that reflects some of these ideas is the Heat-and-run thread migration study [66]. In Chapter 4, I show how to expand upon this by viewing other policies such as DVFS as axes to be used in combination, rather than simply competing policies as assumed in prior works.

1.4 Thesis Overview and Contributions

This thesis is divided into two major divisions. The first part is Chapter 2, which explores power consumption takes for parallel applications on multicore processors with parameter variations. The second part, consisting of Chapters 3 and 4, covers a number of techniques for thermal management.

Chapter 2 explores power management for parallel shared memory applications. Furthermore, an additional design challenge is management of on-chip power variation, which is becoming increasingly significant in deep-submicron technologies. My final results summarize a number of trade-offs with on-off policies with variation-tolerant multicore processors running parallel applications. Due to non-ideal behaviors of scaling some parallel applications, the available tolerance of variation for some applications can be as large as 8.5 W ($\pm$98% of the target per-core design power) on a 32 nm processor consisting of 8 cores.

While Chapter 2 and onwards do not directly take temperatures into account, Chapter 3 introduces thermal management techniques. Using multiprogrammed workloads rather than parallel applications, I propose thermal management techniques for single-core processors with simultaneous multithreading (SMT). While the rest of the thesis explores only multicore architectures, Chapter 3 serves as a precursor to show that similar techniques can be applied even on a single-core processor. I show that by fine-grained control of instruction fetch priorities, the performance of a thermally-stressed uniprocessor can still be increased by 30% while avoiding thermal

emergencies.

Chapter 4 continues the thermal management thrust, and explores a significantly more expansive design space for adaptive thermal management. Rather than a single-core processor I examine thermal management for a four-core processor. A major intent of this chapter is to show how various thermal management techniques work in combination in a multicore environment, so I also model dynamic voltage and frequency scaling (DVFS) combined with other techniques. I explore a fairly exhaustive taxonomy of policies, and find that an overall hybrid policy of dynamic voltage and frequency scaling with sensor-based migration provides the best performance improvement of 2.6X in a heavily thermally stressed environment.

Chapter 2

Power Variation in Multicore Processors Running Parallel Applications

2.1 Introduction

Chip multiprocessors are becoming a widespread basis for platforms in the server, desktop, and mobile sectors. Thus, this chapter, as well as Chapter 4, focuses on multicore designs. Often such processors are used for running several different single-threaded applications simultaneously on one processor. This assumption is used in most prior work, as well as in Chapters 3 and 4 of this thesis. However, it is expected that the software that will run on computing devices will be primarily multithreaded programs within a decade. Thus, this chapter emphasizes heavily on parallel applications.

This chapter formulates power-aware adaptive management techniques for parallel applications running on multicore processors. Furthermore, one of our major contributions is to take into account process variation, a pressing issue for deep submicron

technologies.

Process variations have become increasingly important as deep submicron technologies pose significant risks for wider spread in timing paths and variations in leakage power. Within a few technology generations, it is expected that within-die variations will become more significant than die-to-die variations [7], and manifest in multicore chips as core-to-core variations [36, 37]. Architects must design these chips with appropriate options to ensure reliability while still meeting appropriate performance and power requirements. This involves fallback modes at the circuit, architectural, and system level.

Some post-silicon circuit techniques can be applied to ensure valid timing, but these entail non-trivial costs in terms of dynamic and leakage power. Our starting parameter is $P_{excess,N}$, the amount of excess power on a core resulting from inherent leakage variation. It has been shown that such variations may be significant across multiple cores in a chip multiprocessor [36].

Our work takes into account these power discrepancies to formulate policies for post-silicon adaptivity to meet desired performance and power budgets. Modern platforms often have overall goals or configurable power modes that focus on maximizing the ratio of *performance/watt* rather than performance alone. In studying methods toward this goal, our approach is to turn off cores that are particularly expensive in terms of power consumption. For this it is necessary to establish the appropriate optimal tradeoff points. We seek to quantify these cutoffs depending on the level of variation and execution characteristics of the applications. By giving the system knowledge of power variation traits through diagnostics, these bounds can be known or calculated at system runtime and then used to properly tune performance and power to sustain desired user demands.

Modern devices now run a wide variety of applications, often concurrently, and must efficiently operate depending on their tasks at hand. Thus, we first derive

an appropriate performance/power tradeoff point for the case of running multiple programs simultaneously through multiprogramming. We then extend our analysis to parallel programs, since multicore designs have become a major motivation factor toward seeing widespread use of parallel applications in all sectors.

We propose that multicore-based systems can adapt readily to meet power-performance requirements. Specifically, the core power ratings may be identified at the time of system integration, and can even be reconfigured through system diagnostics after power ratings change due to long-term depreciation effects. With knowledge of these variations, the system can choose how to efficiently allocate cores to particular tasks and put cores to sleep if potential additional performance is not power-efficient. Our specific contributions are as follows:

- We derive an analytical formulation for estimating the amount of tolerable power variation for multicore policies seeking to maximize *performance/watt*. To account for parallel applications, we incorporate Amdahl's Law in this formulation.

- Using Turandot [60] as our simulation platform, we demonstrate these properties using 8 applications from the SPLASH-2 [85] benchmark suite. For example, the high parallel efficiency of `raytrace`, a graphical rendering benchmark, allows it to increase in performance/power ratio by running on as many as 7 cores although this maximum is easily offset by power variation beyond 0.28 W, or about 3.5% of the target per-core design power.

The next section describes our experiment and simulation methodology, while Section 2.3 examines our choice of memory hierarchy on parallel applications. Section 2.4 provides the foundations of our analytical model and without considering multithreaded interactions, while Section 2.5 extends our analysis and provides validation for parallel programs. Section 2.6 extends our studies to possible variation distribu-

tions, while Section 2.7 looks briefly at some basic temperature properties of parallel applications. Finally, Section 2.8 covers related work and Section 2.9 summarizes.

2.2 Experiment Methodology

2.2.1 Architectural Model

We use an enhanced version of Turandot [60] and PowerTimer [8] to model performance and power of an 8-core PowerPC™processor. Our cycle-level simulator also incorporates HotSpot version 2.0 [35, 74] in order to model the temperature-dependence of leakage power from various components. Our process and architectural parameters are given in Table 2.1. Most of the core parameters are similar to those used in recent studies on power management with simulated PowerPC cores [19, 22, 51, 53]. This reflects the recent industry trend of designing CMPs using cores with complexity similar to those in earlier generations (while placing more cores together on a single chip), instead of using wider and more complex cores.

We assume inherent core-to-core power variation across a chip multiprocessor. This can be due to systematic variations which can cause dramatic differences in leakage power across large areas of a chip. The discrepancies of power ratings across these cores may be assumed to take a log-normal distribution [7]. Different power ratings across the cores may also arise even in the context of other adaptive methods. For instance, when target frequencies are not satisfied, a sufficient combination of adaptive body bias (ABB) or adaptive V_{DD} scaling may fix timing variations at the expense of per-core power [80].

2.2.2 Simulation Setup

Our simulation platform is *Parallel Turandot CMP* (PTCMP), our cycle-level simulator. Unlike its predecessor Turandot CMP [51], or its ancestor Turandot, PTCMP is

Global Design Parameters	
Process Technology	$32nm$
Target Supply Voltage	0.9 V
Clock Rate	2.4 GHz
Organization	8-core, shared L2 cache
Core Configuration	
Reservation Stations	Int queue (2x20), FP queue (2x5), Mem queue (2x20)
Functional Units	2 FXU, 2 FPU, 2 LSU, 1 BXU
Physical Registers	80 GPR, 72 FPR, 60 SPR, 32 CCR
Branch Predictor	16K-entry bimodal, gshare, selector
Memory Hierarchy	
L1 Dcache	32 KB, 2-way, 128 byte blocks, 1-cycle latency, 15-cycle snoop latency
L1 Icache	64 KB, 2-way, 128 byte blocks, 1-cycle latency
L2 cache	8 MB, 4-way LRU, 128 byte blocks, 9-cycle latency
Main Memory	80-cycle latency

Table 2.1: Design parameters for modeled 8-core CPU.

programmed with POSIX threads rather than process forking to achieve lightweight synchronization and parallel speedup [20]. This infrastructure is an alternative to Zauber [53], which also avoids slowdown in the fork-based Turandot CMP, but by using a non-cycle-accurate approximation. Our method maintains all necessary cycle-level communication. PTCMP is able to test various combinations of CMP and SMT configurations without core-count limits that have been vexing for some prior simulators [51]. The maintained cycle-level communication not only aids with accurate modeling of shared cache contention, but is also necessary for other enhancements described below.

Like its predecessors, PTCMP also incorporates online calculations for power and temperature through integration with PowerTimer and HotSpot.

2.2.3 Benchmarks

We use 8 of the 12 benchmarks from the SPLASH-2 benchmark suite [85]. Although we have actually conducted experiments with all 12 programs, three of the benchmarks—`ocean`, `fft` and `radix`—are algorithmically restricted from running a number of threads that is not a power of 2. To show clear tradeoffs across the number of cores we have focused only on other benchmarks which do provide this flexibility. Among the remaining 9 benchmarks, `volrend`'s runtimes are an order of magnitude longer than that of other programs, so we have focused on the remaining 8.

Each benchmark was traced using the Amber tool on Mac OS X [3] from the beginning to end of their complete algorithm executions. Amber can generate traces for multithreaded applications, so we traced each benchmark for executions ranging from 1 to 8 threads.

2.2.4 Modeling Thread Synchronization and Coherence

There are two main issues in our extensions to Turandot for parallel program simulation: synchronization and coherency. Fortunately, from an implementation perspective these can be dealt with independently. We implement modeled lock synchronization (not to be confused with the implementation's internal synchronization) and a MESI cache coherence protocol [62] to maintain memory consistency across each core's local cache with respect to the shared L2 cache.

We use Amber's thread synchronization tracing system in order to accurately track the status of pthread-based mutexes and condition variables. Our trace-driven simulator then models stalls for individual threads when such thread-communication dependencies are detected.

For shared memory coherence we implement a MESI [62] cache-coherence protocol to allow private copies of data in each core's local data cache with respect to the shared L2 cache. We have correspondingly extended PowerTimer to account for the energy

cost of cache snoop traffic.

Although this simulation approach is capable of simulating parallel applications, it remains entirely trace-driven. Thus, it can be assumed that it has some limitations in terms of simulation accuracy compared to execution-driven methods. Although we have not performed a complete validation study for this trace-driven technique, one initial observation is that the various SPLASH-2 applications simulated on this infrastructure have shown performance scaling properties similar to those observed by Li et al. [47] using an entirely execution-driven simulation infrastructure. For a more extensive study, one possible method for validation of the trace-driven approach would be to link Turandot with an execution-driven parallel application simulator and compare the simulation results of the hybrid approach against the fully trace-driven approach.

2.3 Consequences of the Memory Hierarchy

The bulk of experiments in this chapter assume an 8 MB shared cache, which is roughly an expected size for an 8-core processor built on 32 nm technology [39]. However, this was not done with any evidence that the applications under study require such a large shared cache. In fact, using such a large shared storage most likely avoids any capacity misses that would be common on the original systems for which SPLASH-2 was designed.

Figures 2.1 through 2.8 shows the performance effect of smaller shared cache sizes on `barnes`. As shown in each graph, the smallest test case is 128 kB, which is already extremely unrealistic for an 8-core processor with local caches of nearly the same size. Even in this case, for `barnes` and `cholesky` there is at most 5% performance degradation as compared to the 8 MB cache, and less for other applications. This result may reflect the datedness of SPLASH-2 applications, which were designed for

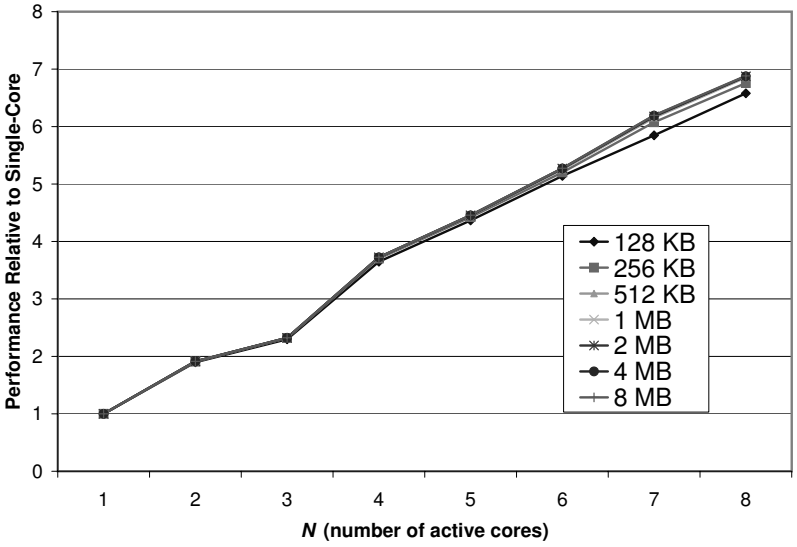


Figure 2.1: Performance of `barnes` across different shared L2 cache sizes.

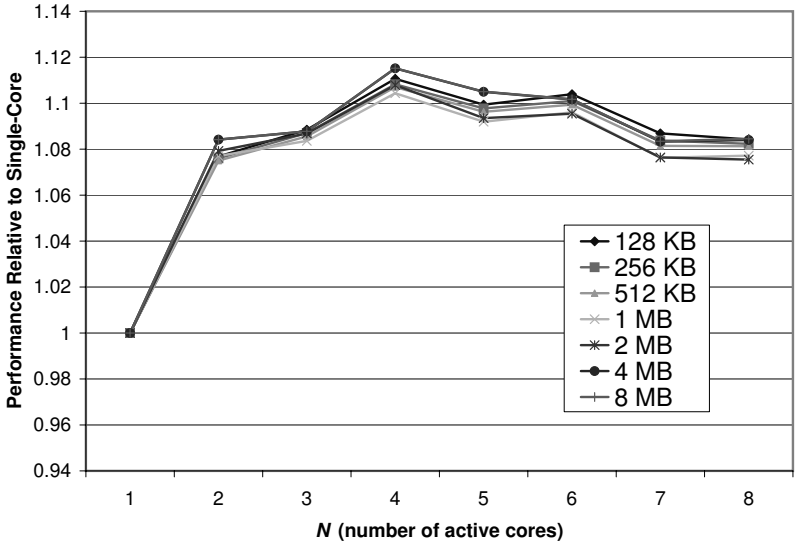


Figure 2.2: Performance of `cholesky` (including the long initialization phase) across different shared L2 cache sizes.

shared-memory processors which did not have any large global shared storage.

Even though L2 cache sizes do not significantly affect the performance of these benchmarks, smaller L2 caches result in a significant portion of leakage power savings. Thus, it can be expected that the smaller L2 cache configurations may provide better performance/watt ratings. An example of this tradeoff is shown with the example of

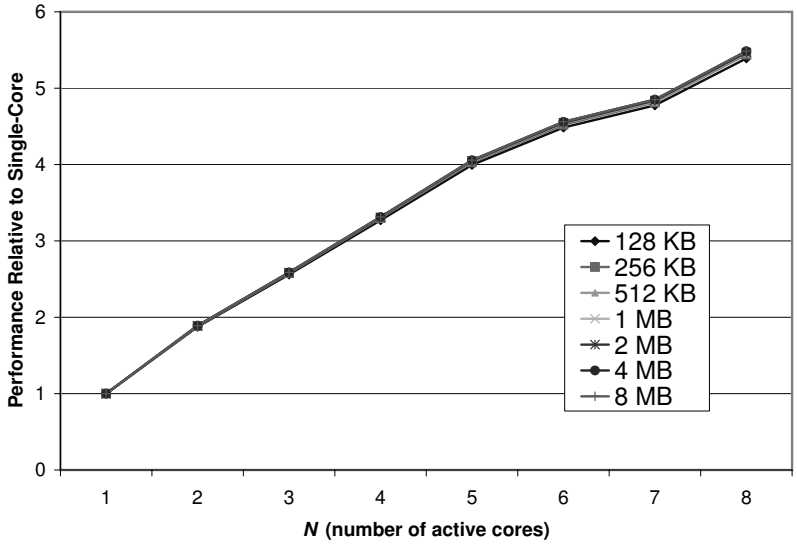


Figure 2.3: Performance of `fmm` across different shared L2 cache sizes.

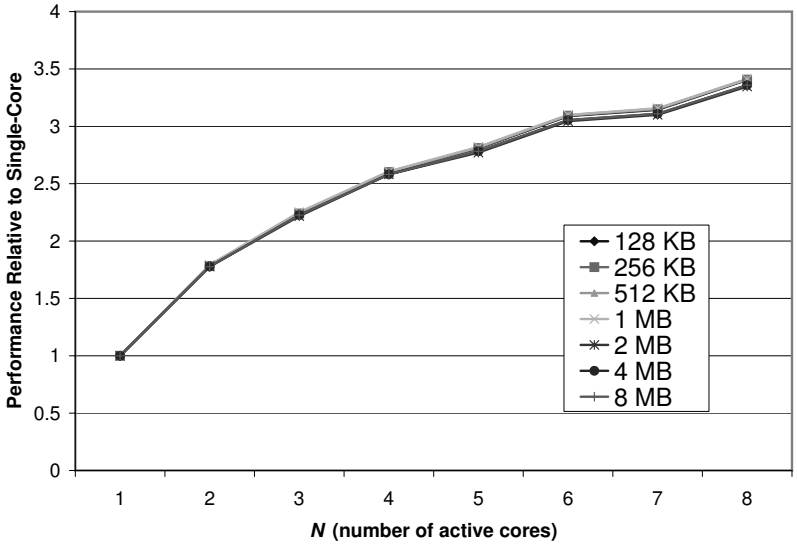


Figure 2.4: Performance of `lu` across different shared L2 cache sizes.

`barnes` in Figure 2.9. In this example, the configurations with L2 caches of 2 MB or less find the most power efficient runtime configuration to be at most 5 cores. With a 128 KB cache, the best performance/watt rating for `barnes` is reduced to the single-node case of $N = 1$. This result may be expected for compute-intensive benchmarks that do not require significant amounts of memory communication, as SPLASH-2 has

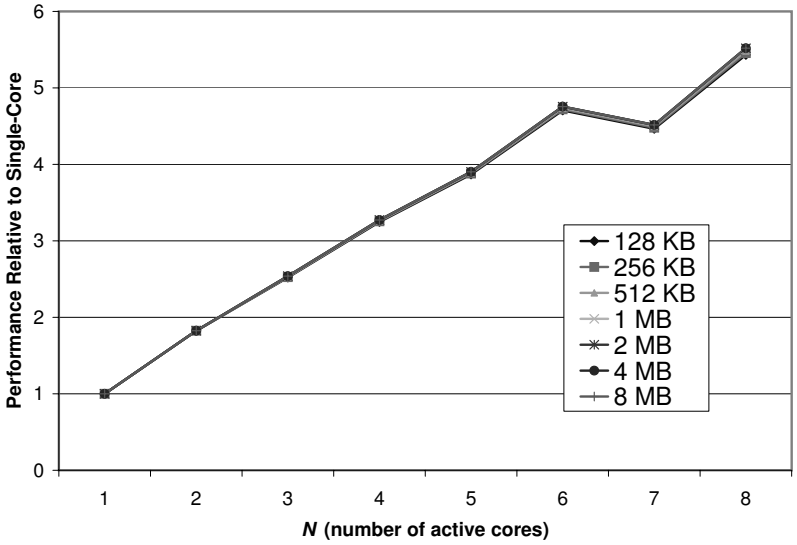


Figure 2.5: Performance of `radiosity` across different shared L2 cache sizes.

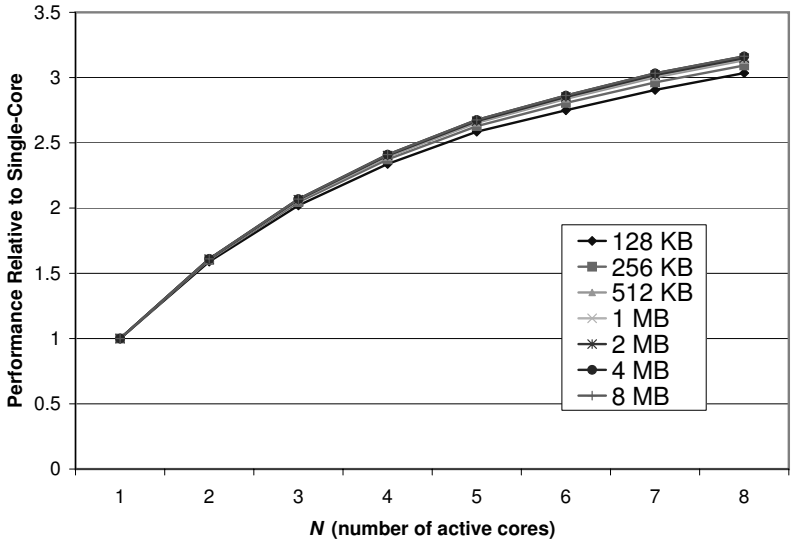


Figure 2.6: Performance of `raytrace` across different shared L2 cache sizes.

been written for. For such benchmarks, the extra cache capacity results mostly in wasted leakage power.

One effect known to be possible in some scenarios at small L2 caches sizes was super-linear speedup. Since the early days of shared virtual memory, this effect was prominent due to additional nodes supplying more capacity [50]. This is certainly

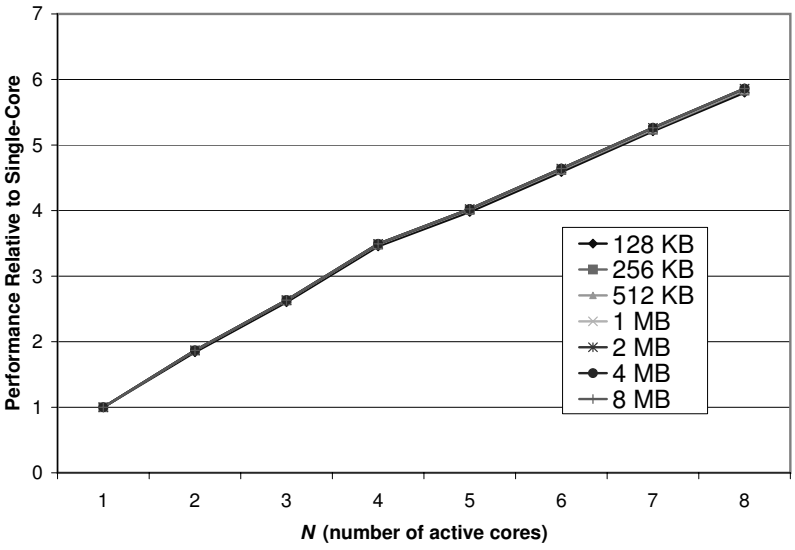


Figure 2.7: Performance of `water-nsquared` across different shared L2 cache sizes.

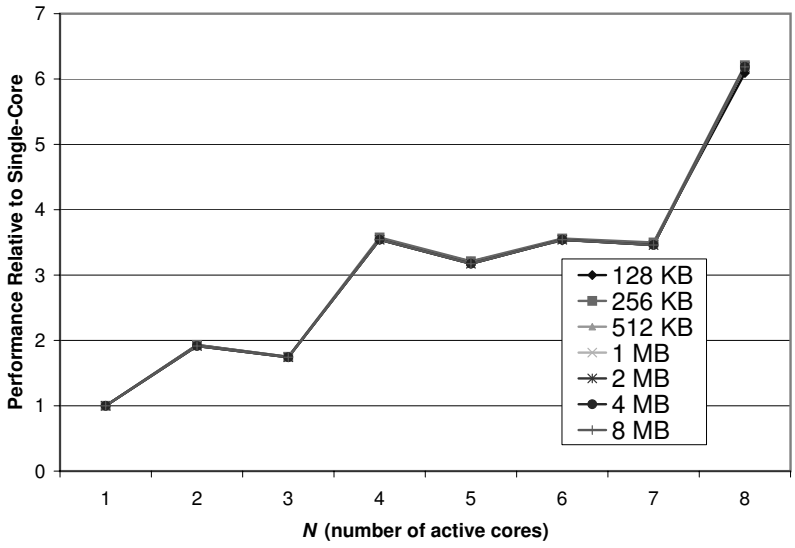


Figure 2.8: Performance of `water-spatial` across different shared L2 cache sizes.

not the case in Figures 2.1 through 2.8. Such an effect would have manifested as significant performance degradations in the single-node case with less performance degradation at higher values of N. However, we obtain only similar small performance degradations in all levels of parallelization.

The section has shown that, for these particular benchmarks, smaller cache sizes

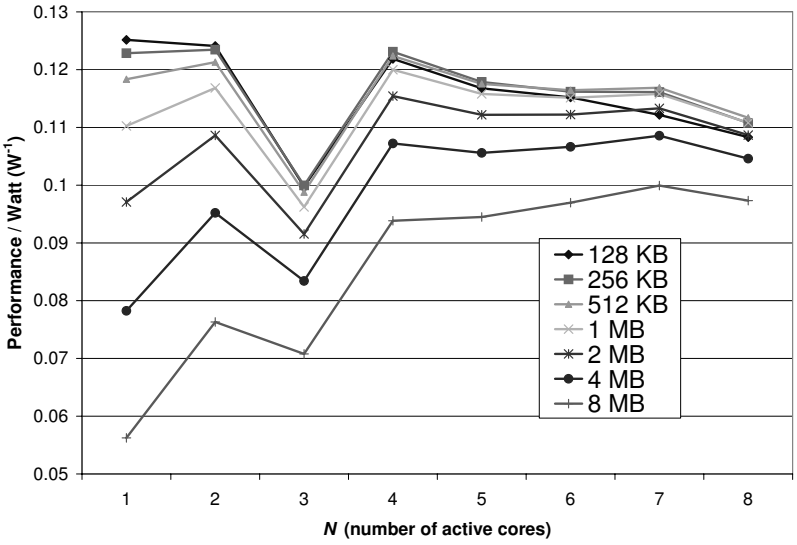


Figure 2.9: Performance/watt ratings for `barnes` across different shared L2 cache sizes.

may be beneficial for power savings without adversely impacting performance. Although the experiments in the remainder of this chapter all assume an 8 MB shared cache, it should be remembered that the performance/watt results given in the various sections could potentially be improved by assuming a smaller shared cache.

2.3.1 Metrics

As a primary metric, this chapter assumes the goal of maximizing the ratio of performance per watt. At the very least, this requires some sort of performance rating. For parallel applications, a practical performance rating is the speedup ratio relative to the performance of a single node. This is then divided by the total power across all cores and shared resources to obtain a performance/watt rating in units of W^{-1}.

2.4 Basic Analysis

2.4.1 Formulations for Multiprogrammed Workloads

We seek to maximize the throughput/energy ratio in spite of excess power on cores due to process variation, defined as $P_{excess,N}$. This excess power can arise due to inherent leakage variation due to systematic variations in lithography. In a more complex scenario this can arise as the after-effect of some post-circuit tuning. Specifically, cores that do not meet timing requirements at the time of manufacture can be receive ABB or V_{DD} adjustments, but this may cause these cores to go beyond their specified power limit [80].

ABB, when applied in forward mode, involves placing a positive bias between the body and source. Thus, these two techniques may be best used in combination to ensure timing requirements. This does not increase the dynamic switching power, but increases leakage power significantly more than V_{DD} adjustment [80]. Thus, these techniques may be best used in combination to ensure timing requirements.

For a given timing adjustment the supply voltage must be scaled up roughly linearly, resulting in an approximately linear increase in leakage power and quadratic increase in dynamic power. For cores which already meet their timing requirements with sufficient slack, we may also apply ABB in reverse (known as reverse body bias, RBB) or lower V_{DD} in order to save power. Even in a fortunate scenario where all cores have some timing slack, the degree to which ABB or V_{DD} adjustment can be applied will differ across cores. This combined with inherent leakage variation results in a set of cores on one die with possibly very different power characteristics.

For the purpose of managing these resultant power variations, the metric we aim to maximize is the ratio of *performance/watt*, a current focus for modern server applications [45] and one of the primary concerns for mobile platforms. We have also considered some more complex scenarios such as minimizing power for a fixed per-

formance deadline or maximizing performance for a fixed power budget. These other analysis routes are interesting areas for future study, but for simplicity we have chosen to focus on maximizing the performance/power ratio.

Our approach is to find the appropriate cutoff point such that a system may decide to turn a power-hungry core off. There may often be a benefit to retaining an extra power-hungry core, since more running cores can help amortize power cost of dynamic and leakage power from shared resources such as the L2 cache. In addition to the cache and communication buses, multicore processors typically have many other elements shared among the cores. For example, in a multiple-socket system, there may be inter-socket links. These can can be significantly power consuming, yet required to constantly negotiate between all cores and thus are difficult to shut down. The variable P_{shar}, used in various equations throughout this chapter, represents the total power cost of such resources.

We wish to see the appropriate cutoff point for when this core offers enough performance to make its wattage worthwhile, versus when we should put this core into sleep mode and make do with the remaining resources. Our criteria for when a core should be disabled can be stated in terms of an inequality relating the performance/power ratio of N cores to $N-1$ cores, as follows:

$$\frac{perf_N}{power_{N\&excess}} \leq \frac{perf_{N-1}}{power_{N-1}} \tag{2.1}$$

Here, $perf_N$ and $power_{N\&excess}$ represent the performance of power of the full processor with all N cores used, including the core with excess power. The corresponding $perf_{N-1}$ and $power_{N-1}$ represent those values when the Nth core (sorted from lowest to highest excess power) is turned off. We then expand the condition of Equation

(2.1) as such:

$$\frac{N perf_1 \alpha_N}{N P_{core} + P_{excess,N} + P_{shar,N}} \leq \frac{(N-1) perf_1 \alpha_{N-1}}{(N-1) P_{core} + P_{shar,N-1}} \tag{2.2}$$

where N represents the number of active cores, P_{core} denotes average core power, P_{shar} denotes power shared among cores, such as power consumed by the L2 cache, and α represents a speed factor to take resource contention into account. Various elements such as P_{shar} are subscripted to indicate they have a specific value for different core configurations, while others such as P_{core} are taken as constant across different values of N. In fact we assume only a single value of P_{core}, since core power tends to vary significantly less than cache and interconnect power.

The α parameter represents the slowdown caused by contention on shared resources, a critical design element of CMPs. It is a factor typically less than 1. If there is little shared cache or memory contention, it becomes likely that $\alpha \approx 1$ [51]. This property does not hold for memory-intensive benchmarks, so we use the much weaker assumption that $\frac{\alpha_N}{\alpha_{N-1}} \approx 1$, which says that the incremental contention from an additional core is reasonably small for moderately sized N.

Solving for $P_{excess,N}$ under these conditions, this results in:

$$P_{excess,N} \geq (\frac{N}{N-1}) P_{shar,N-1} - P_{shar,N} \tag{2.3}$$

This equation states our criterion in a relatively simple manner, by depending only on the power cost of shared resources but not the baseline core power nor contention factors. In essence, we plan to turn off any cores for which Equation (2.3) is true.

A key limitation of Equations (2.2) and (2.3) is that these do not take into account interactions and bottlenecks between the different threads. For a multiprogrammed workload consisting of single-threaded applications, this is a reasonable assumption. In [21], we even demonstrated how this criterion usually holds for multiprogrammed

workloads consisting of SPEC 2000 benchmarks.

If the condition of Equation (2.3) were to be checked and acted upon dynamically, this would require knowing the value of $P_{excess,N}$, which is calculated per core relative to the average across all cores, and P_{shar}. Direct power measurement, such as used in Intel's Foxton technology [61], could be done individually on all cores and the shared cache in order to provide the numerical input for these calculations.

While not readily apparent from Equation (2.3), a general characteristic of the cutoff point is that it increases roughly linearly with respect to the power cost of shared resources. This can be seen more clearly in the simplest case. When P_{shar} does not vary significantly with respect to N, the expression in Equation (2.3) simplifies down to:

$$P_{excess,N} \geq \frac{P_{shar}}{N-1} \tag{2.4}$$

In this case, the amount of room for power variation increases linearly with P_{shar}. Similarly, with N in the denominator, increasing the number of active cores takes care of amortizing these costs and hence reduces room for large values of $P_{excess,N}$.

2.4.2 Underlying Themes

Our formulated bounds focus only on CPU power, but system designers may wish to include the full power cost of other resources including RAM, chipsets, memory-buffering add-ons, and other essentials. Equation (2.4) describes the general trend of how including all these elements generally serves to raise the cutoff point, due to better amortizing shared costs. This is one way to confirm general intuition regarding power efficiency of multicore designs.

On the flip side, Equation (2.4) also denotes an inverse relation between the $P_{excess,N}$ cutoff and the number of cores for maximizing performance/watt. In the future, if multicore designs can feasibly scale up to many more cores, this increases

the chance that one core may become no longer worthwhile to use in the goal of maximizing performance/power. Even so, modern mobile platforms are designed with multiple power modes, and feasibly a highest power mode may seek to maximize performance and still have use for the core in some situations, while a lower power mode may aim to maximize performance/watt as has been the goal in our work.

The relations discussed in this section give an intuitive view of the tradeoffs in turning off cores for power savings. Nonetheless, one limitation is that these do not take into account the sequential bottlenecks of parallel applications. In the following section we remedy this by extending the formulation with Amdahl's Law.

2.5 Analysis and Results for Parallel Applications

2.5.1 Extending the Basic Analysis with Amdahl's Law

Our analysis here uses much of the same methodology as in Section 2.4.1. A key difference is that our speed factor is no longer $\alpha_N N$, which is effectively linear, but rather dictated strongly by Amdahl's Law for parallel computation. The vast topic of parallel computation can certainly entail many complex versions of Amdahl's Law [30, 56, 71], but we choose the most basic form as sufficient for our analysis:

$$speedup_N = \frac{1}{s + \frac{1-s}{N}} \tag{2.5}$$

where s represents the fraction of sequential/serial computation that cannot be parallelized, and likewise $(1 - s)$ represents that fraction that can be parallelized. The value of s is an important application characteristic in deciding optimal performance tradeoffs. Using this, we perform an analysis similar to that used in Section 2.4.1.

We evaluate Equation (2.1) and solve for $P_{excess,N}$ as follows:

$$\frac{\frac{1}{s+\frac{1-s}{N}}perf_1}{NP_{core} + P_{excess,N} + P_{shar,N}} \leq \frac{\frac{1}{s+\frac{1-s}{N-1}}perf_1}{(N-1)P_{core} + P_{shar,N-1}} \tag{2.6}$$

$$P_{excess,N} \geq \frac{s+\frac{1-s}{N-1}}{s+\frac{1-s}{N}}[(N-1)P_{core} + P_{shar,N-1}] - NP_{core} - P_{shar,N} \tag{2.7}$$

The above relation is more complex than the properties found in Section 2.4, but we can use it to study several properties of parallel programs. Because this criterion relies heavily on s, which is an empirical constant that varies not only from benchmark to benchmark but even within different configurations for a single benchmark, it cannot be used to accurately predict cutoff points. It does, however, have distinct limit properties that give us much insight to the general power behavior of parallel applications.

First, Equation (2.7) is actually a special case of its analog for the multiprogram analysis. When $s = 0$, this represents that the application is completely parallelizable with no penalty of dependencies or contention. Substituting $s = 0$ into the expression and simplifying yields exactly Equation (2.3).

On the other hand, the case of $s = 1$ represents the worst case of limited parallel speedup, where a program will not run any faster on multiple cores as compared to just one. If we substitute in $s = 1$, the expression evaluates to $P_{excess,N} \geq -P_{core}$, which is always true and confirms that in such a specific situation reducing the execution down to a single core will always increase the performance/power ratio. Thus, in order to seek a $P_{excess,N}$ cutoff that is greater than zero, it helps if $s \ll 1$.

Next, always of interest in parallel programming problems are the limits of scaling up to large values of N. Our goal is to maximize performance/watt, so a suboptimality condition would result when turning off one of the cores would improve the performance/watt ratio. Taking the limit for large N with a nonzero value of s results in

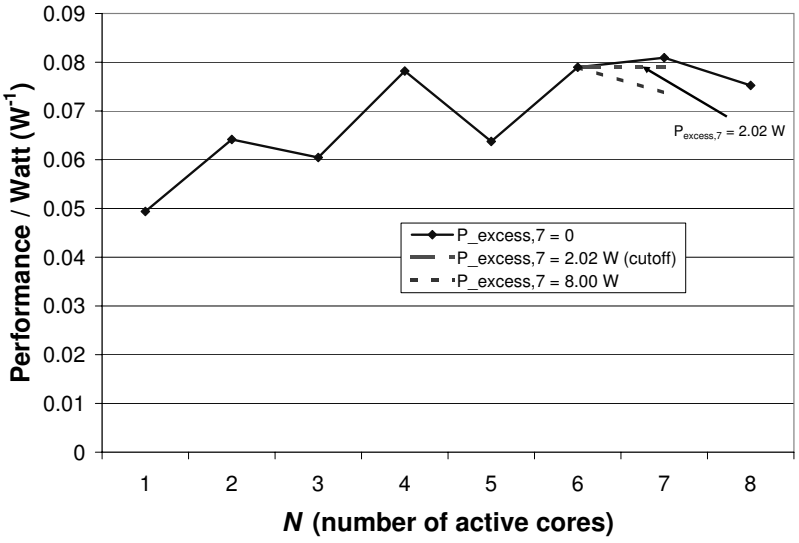


Figure 2.10: Performance/watt ratios for `barnes` across varying N, and varying $P_{excess,N}$ at $N = 7$.

the following suboptimality condition:

$$P_{excess,N} \geq P_{shar,N-1} - P_{shar,N} - P_{core} \tag{2.8}$$

Since typically $P_{shar,N} \geq P_{shar,N-1}$ this relation almost always also evaluates to a negative cutoff value for $P_{excess,N}$, meaning that the performance/power ratio only decreases after going beyond some finite N. Intuitively, we would thus expect a plot of this ratio vs increasing N to be a concave-down function that begins to decrease after leveling off. Furthermore, if the performance/power ratio grows only slowly before reaching this peak, we would expect a smaller allowable range for $P_{excess,N}$, as compared to the relation found in Section 2.4 where the ratio would continue growing regardless of the size of N. The following results in Section 2.5.2 confirm these limit behaviors.

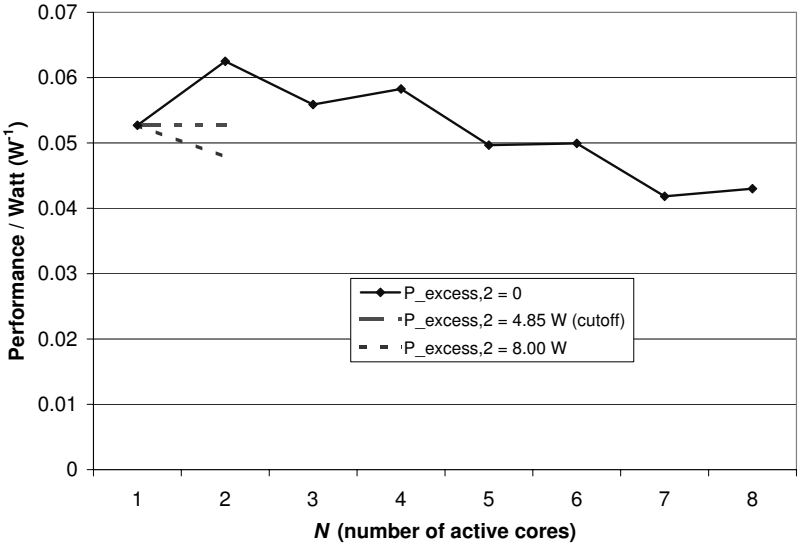


Figure 2.11: Performance/watt ratios for `cholesky` across varying N, and varying $P_{excess,N}$ at $N = 2$.

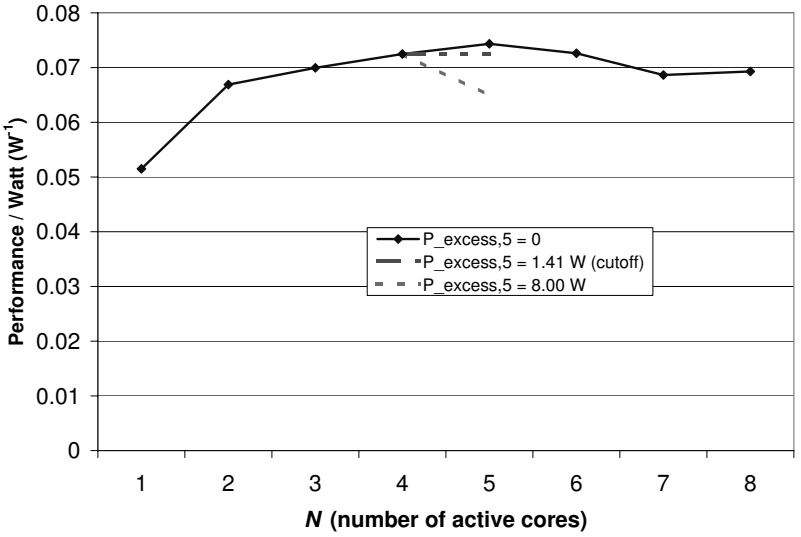


Figure 2.12: Performance/watt ratios for `fmm` across varying N, and varying $P_{excess,N}$ at $N = 5$.

2.5.2 Experimental Results

Although we use full traces as described in Section 2.2, we analyze only the true execution phase of each SPLASH-2 benchmark run for our timing and power measurements. This phase begins after creation of all child threads and ends upon their

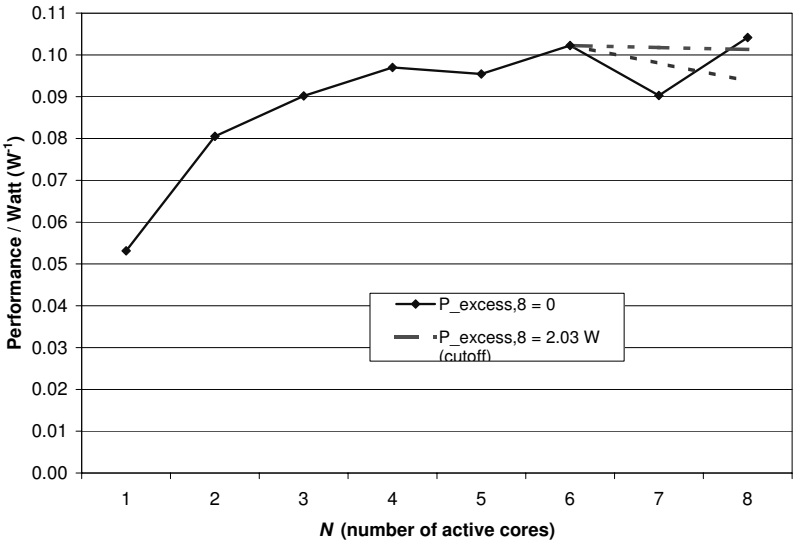


Figure 2.13: Performance/watt ratios for `lu` across varying N, and varying $P_{excess,N}$ at $N = 8$.

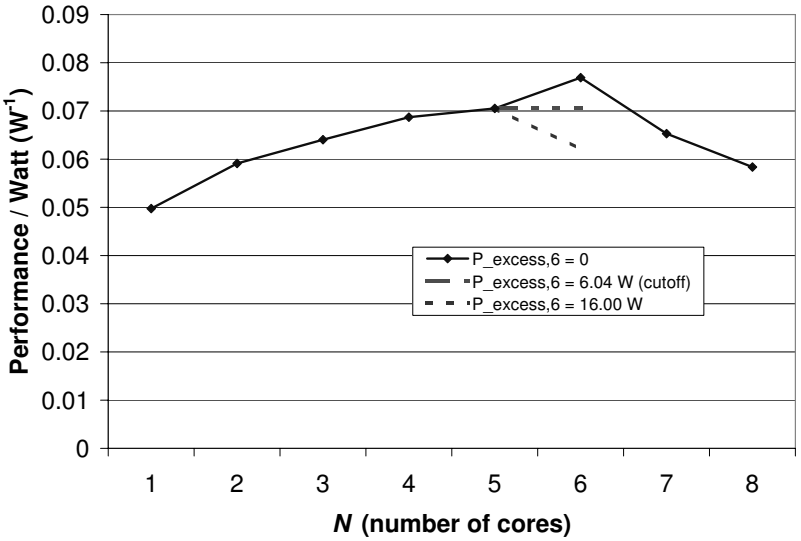


Figure 2.14: Performance/watt ratios for `radiosity` across varying N, and varying $P_{excess,N}$ at $N = 8$.

completion, but does not include any long initialization phases beforehand nor the section of code at the end of each benchmark that generates a summary report.

Figures 2.10 through 2.17 give examples of varying performance/power ratios with respect to N and $P_{excess,N}$. In each case, the main curve spanning all core counts as-

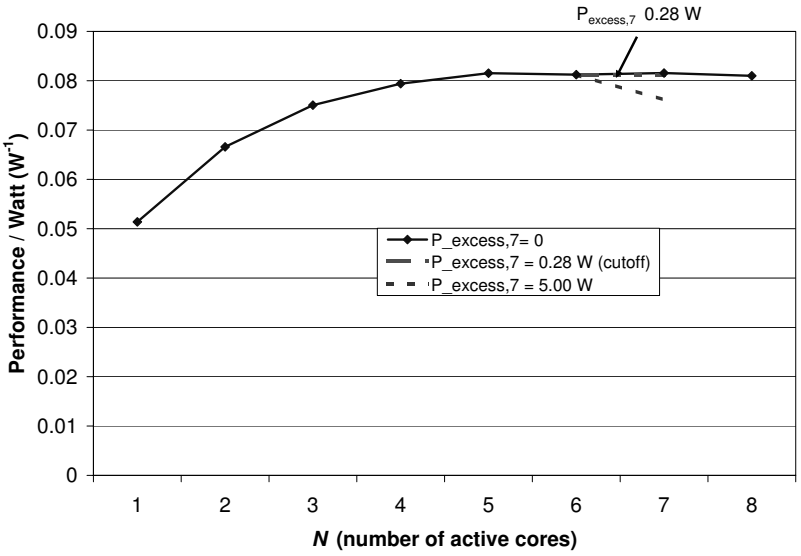


Figure 2.15: Performance/watt ratios for `raytrace` across varying N, and varying $P_{excess,N}$ at $N = 6$.

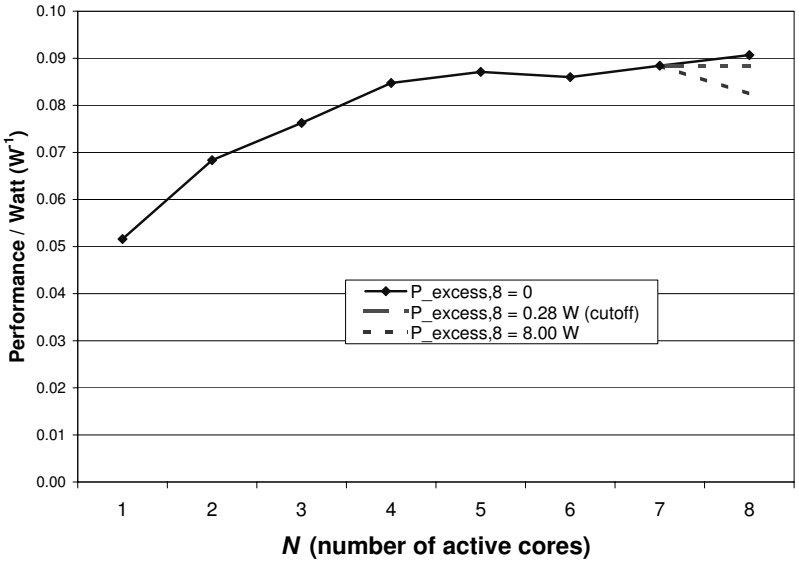


Figure 2.16: Performance/watt ratios for `water-nsquared` across varying N, and varying $P_{excess,N}$ at $N = 7$.

sumes zero power excess on all cores. The two additional lines represent the change in power efficiency for possible values of $P_{excess,N}$ at the otherwise optimal performance/watt point. For example, in `raytrace`, we see a best configuration at $N = 7$, with only a small allowable range of power variation. On the other hand, in `cholesky`,

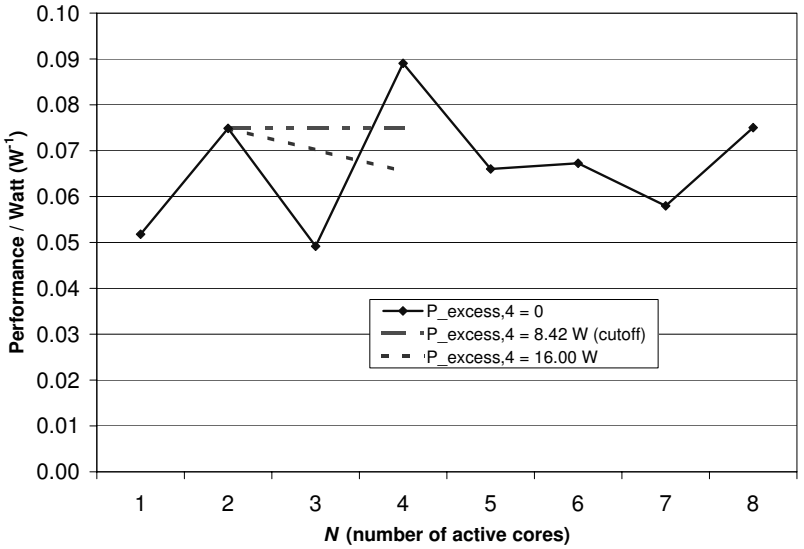


Figure 2.17: Performance/watt ratios for `water-spatial` across varying N, and varying $P_{excess,N}$ at $N = 4$.

we see good power efficiency occurring not beyond 2 cores. This is largely in part due to this algorithm's large serial portion [85].

The non-smooth patterns in the `cholesky` graph reveal other notable effects. In particular, core configurations that are not set as a power of 2 each take an additional performance reduction, consequently resulting in poorer performance/power. In fact, most of our benchmarks were found to show some degree of performance preference towards power-of-2 thread counts. This can be explained by non-ideal realities such as cache alignment. Such additional factors affecting performance have a rather direct effect on the performance/power ratio.

One notable result, that would not be predicted in the simplified analysis for multiprogrammed workloads in Section 2.4, is that limitations are reached at a finite number of cores, as formulated by our analysis in Section 2.5.1. Our results for all parallel programs are summarized in Table 2.2. Individual s values used in our calculations for various benchmarks are calculated from the best fit according to speedups obtained through our simulations. However, for we have also tested some

programs for limited thread counts on a real 8-way SMP system to confirm similar respective parallel speedup characteristics. For example, `fmm` which is easier to observe than some other benchmarks because it does not have a large initialization phase, showed near linear speedups with one to three threads on Solaris.

There are a few interesting cases shown in Table 2.2. For one, `lu`'s characteristics best fit a negative value of s, meaning it received super-linear speedup with respect to N in some cases. This is unusual, although not impossible, as many complex effects such as improved cache hit ratios can combine for such a result.

Second, `radiosity` and `water-spatial` have unusually high allowable $P_{excess,N}$ ranges for their optimal core configurations. The reason for this seems to be due to an interplay of effects that affect adjacent configurations extraordinarily negatively. For example, `radiosity` shows some unexpected performance degradation when running on five cores, which makes $N = 6$ a particularly better choice with a large $P_{excess,6}$ range. Similarly, `water-spatial` shows unusually low performance/watt at three cores, making $N = 4$ a very stable point for maximizing performance/watt.

Overall, these results confirm much of the intuitive limit behavior specified by our analytical formulation. However, this formulation cannot accurately predict the numerical value of the $P_{excess,N}$ cutoff at any given finite point. For example, although `raytrace` matches shows an appropriate small $P_{excess,N}$ cutoff of 0.28 W (about 3.5% of the target per-core design power) `water-spatial` shows an unexpectedly large cutoff of 8.42 W (98% of the target per-core design power). These deviations from the analytical prediction are due to many application-specific non-ideal factors. A power-efficient multicore system design can take advantage of estimates based on the limit behaviors we have formulated, but an interesting topic for future study would be how a dynamic policy would adjust estimates to take into account application-specific special cases. Such a policy would not only utilize direct measurement of core power, as needed in the multiprogrammed case of Section 2.4, but also involve performance

application	s	max N	$P_{excess,(maxN)}$ cutoff
barnes	0.039	7	2.02 W
cholesky	0.254	2	4.85 W
fmm	0.066	5	1.41 W
lu	-0.009	8	1.34 W
radiosity	0.083	6	6.04 W
raytrace	0.044	7	0.28 W
water-nsquared	0.025	8	2.03 W
water-spatial	0.019	4	8.42 W

Table 2.2: Experimental results for SPLASH-2 benchmarks, showing most power-efficient N, cutoff for $P_{excess,N}$ at that configuration, and each benchmark's corresponding s value.

monitoring to track parallel efficiencies.

2.6 Mapping to Log-Normal Distributions

Although the $P_{excess,N}$ model is applicable to any distribution, one shortcoming is that it gives little intuition as to the magnitude of core-to-core variations. Although it provides a bound for a particular configuration, in order to make use of such a model a user must be aware of how $P_{excess,N}$ changes when starting from a different number of active cores.

Another more fundamental, possibly more intuitive, way of viewing variation is to map these on to an actual variation profile. Because a distribution of inherent leakage variation has been shown to be log-normal [7], we use samples of log-normal distributions. However, because leakage power actually results from many other factors including random and systematic variations in different processing technologies, this result is not universal. The distribution may also become much more complicated than simply log-normal when other techniques such as adaptive ABB or voltage scaling are involved. Although not as generalizable as the $P_{excess,N}$ formulation, mapping to these specific log-normal distributions can perhaps provide strong intuition.

A log-normal distribution is a probabilistic distribution, so to fully view a wide range of possibilities we would require Monte Carlo simulation. To simplify this problem, although at the loss of generality, we assume simplified distributions formed from integrating under a log-normal probability distribution. With this technique, our main tuning parameter becomes σ/μ, which represents the steepness of the log-normal distribution. σ represents the standard deviation while μ represents the mean of the distribution. Since these quantities are only meant to serve as exponent values, μ is set to 1. Figure 2.18 shows the performance/watt ratings for `barnes` across log-normal distributions with varying σ/μ.

In order to obtain a discrete (sampled) log-normal distribution for each value of σ/μ, rather than resorting to Monte Carlo simulation, we integrate underneath the continuous Gaussian curve to obtain 8 points that evenly distribute the cumulative distribution. This effectively causes points in the center of the discrete sample to bundle close together (the middle of the bell curve having higher probability density) while the outlier points near the beginning and the end are spaced further apart (with lower probability densities requiring more length integrated underneath the curve in order to evenly partition the cumulative distribution between points). In order to convert this normal distribution into a log-normal distribution, each sample is then exponentiated, using a base of e, to obtain the log-normal distribution assuming the geometric mean to be exactly between the 4th and 5th cores. This means that the 4th and 5th cores will have effectively the least variation, while the 8th core will have the largest power increase and the 1st core will have the largest power decrease.

We chose the various values of σ/μ to give a large range of leakage variation. For perspective, a value of $\sigma/\mu = 0.010$ forms a distribution where the 8th core is 2.5X as power-consuming as the first core, while a value of $\sigma/\mu = 0.050$ results in the 8th core being 100X as power-consuming. In some results based on real test chips, Borkar et al. showed a 30X deviation in leakage power across a set of hundreds of chips [7].

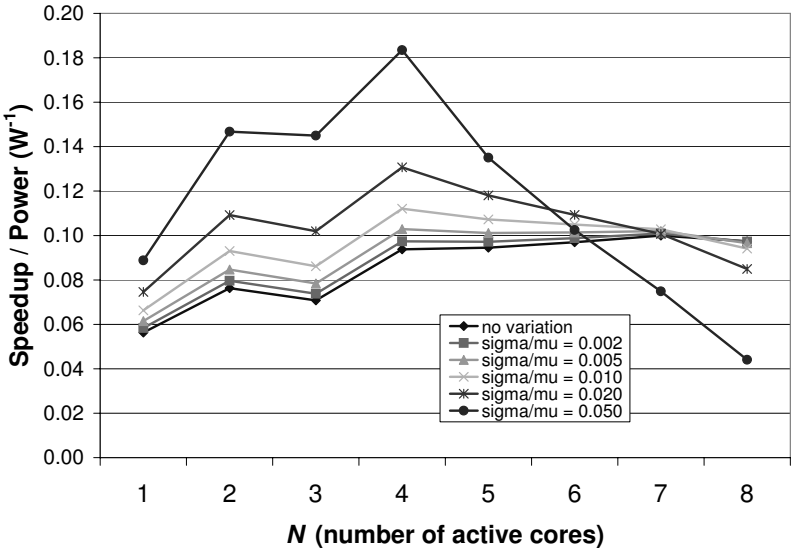


Figure 2.18: Performance/watt ratings for `barnes` across log-normal distributions of various σ/μ values.

These curves find that although a σ/μ setting of 0.002 is not sufficient to change the best configuration away from $N = 7$, that is achieved at $\sigma/\mu = 0.005$. At $\sigma/\mu = 0.005$, the best performance/watt ratio is found at $N = 4$, and this remains the ideal configuration even for steeper log-normal distributions. The suboptimality of many active cores is an expected result for steep distributions which have very power-hungry tail-end cores.

This does not quite explain, however, why the best configuration does not worsen to fewer cores than $N = 4$ even for steeper distributions. This turns out to be a direct result of the shared L2 cache overhead power. The greater values of σ/μ result in relatively low-power cores (compared to the mean core power set between the 4th and 5th cores), and the performance improvement provided by each of these bottom four cores improves the performance/watt ratio.

2.7 Temperature Properties of Parallel Applications

Although this chapter uses performance and power as our key metrics, aggregate power does not necessarily translate directly into aggregate temperature. By contrast, Chapters 3 and onward of this thesis motivate and demonstrate simulations focusing on temperature control rather than power management. To compare and foreshadow how temperature connects with the experiments in this chapter, in this section we examine some thermal properties of the 8 parallel applications. Because a thermal simulator, HotSpot [35, 74], is incorporated into our simulation infrastructure, we are able to gauge the thermal properties of parallel applications from our simulations. In earlier work by Huang et al., HotSpot 2.0 was validated against a thermal test chip [35]. When tied to the PowerPC architecture simulation, HotSpot has been used for various experiments in dynamic power management [19, 22, 51, 53]. These studies have required HotSpot's steady-state temperature approximation as well as transient temperature calculations. In this section, we use only the steady-state temperature option.

Unlike Section 2.6, in this section we do not map a distribution of variations onto the various cores, nor do we assume various $P_{excess,N}$ values on the last core as done in earlier sections. All simulations in this section represent zero built-in core-to-core power variation, and thus form an overview of the inherent application-level thermal behavior alone. These inter-thread thermal distributions provide a base for comparison to the inter-application thermal distributions demonstrated through real measurements in Chapter 1 and exploited in Chapters 3 and 4.

Table 2.3 lists the various peak temperatures obtained using the steady-state temperature approximation of HotSpot [74] for all eight applications running our test processor. Across the entire chip, the peak temperature spot was found to be at the

application	"hot core" hotspot temperature	"hot core #2" hotspot temperature	"cold core" hotspot temperature
barnes	91.97 °C	91.80 °C	91.17 °C
cholesky	88.05 °C	81.78 °C	81.64 °C
fmm	85.02 °C	84.58 °C	84.07 °C
lu	91.86 °C	82.51 °C	82.43 °C
radiosity	89.06 °C	85.94 °C	85.43 °C
raytrace	86.92 °C	82.26 °C	82.15 °C
water-nsquared	85.90 °C	84.80 °C	84.56 °C
water-spatial	86.48 °C	85.15 °C	85.14 °C

Table 2.3: Temperatures of key hotspots when running all 8 threads for various benchmarks. The "hot core" temperature is the hottest location on the entire chip, while "hot core #2" shows the hotspot temperature of the second-hottest core, while the "cold core" temperature is the hotspot temperature of the least hot core.

fixed-point execution unit (FXU) register file on one of the cores. Several of these sampled temperatures are shown in Table 2.3. As a whole, this subset of temperature values represents only the small hotspots of the chip, whereas larger areas, such as the L2 cache, may be more than 15 °C lower than the FXU register file temperatures.

The "hot core" hotspot temperature for each benchmark was found to always appear on the core used for the initialization phase of each benchmark. On some of our benchmarks with a relatively long initialization, such as `lu`, this can result in a huge difference between the hottest core and the second-hottest core. Despite this asymmetry, since all of the remaining seven cores are not used for initialization, the difference between the second-hottest core and the least warm core is strikingly small. Across all eight benchmarks, this difference is never more than 1°.

This brief thermal experiment does confirm that there are different thermal properties among different parallel applications. However, another observation is that the inter-core thermal variations may not be significant within any one application. SPLASH-2 applications are are not known to have significant heterogeneous activity, and other parallel applications may show similar properties. For this reason, in

Chapters 3 and 4 we focus on multiprogrammed workloads, instead of parallel applications, which show much thermal variation within and are thus a prime candidate for adaptive management.

2.8 Related Work

Much prior work has examined power and performance characteristics of multicore architectures when running multiprogrammed workloads [18, 27, 42, 44, 51, 53, 66] as well as parallel applications [4, 23, 41, 48, 49]. To the best of our knowledge, however, ours is the first and at this time still the only to examine parallel applications in the context of process variation.

Although the majority of past research on variation tolerance has been at the circuit and device levels, recently a number of architectural approaches have been proposed. These include variation-tolerant register files [54], execution-units [55], [2, 58], and pipeline organizations [25, 83]. Furthermore, Humenay et al. propose a model for variations in multicore architectures [36, 37] while Chandra et al. provide a methodology for modeling variations during system-level power analysis [12].

2.9 Summary

Our work presents a foundation for power-performance optimization in the face of process variation challenges. We have formulated a simple analytical condition relating the shared power costs to predict an optimal cutoff point for turning off extra cores. Using PTCMP to model an 8-core processor, we have analyzed power variance bounds for 8 of the SPLASH-2 benchmarks. Parallel applications such as these have become increasingly relevant as software in the server, desktop, and mobile sectors all move toward more common use of multithreaded applications. In augmenting our equations with Amdahl's Law, we use the parameter s to represent each application's

fraction of sequential execution, and have formulated a model to predict limit property trends across several parameters and demonstrated these accordingly with the SPLASH-2 benchmarks. For further intuition, we map our simulation results onto possible log-normal distributions, explore the impact of the memory hierarchy, and present an overview of the temperature properties of these applications.

The purpose for finding these appropriate tradeoff points comes from a system design perspective. If a system is aware of its inter-component dynamic and leakage power excesses, it can make variation-aware decisions for allocating cores in a power-efficient manner. In an age when portable devices may execute different types of applications, such techniques are necessary to provide appropriate tradeoffs in performance and power in spite of different application characteristics and process variations.

Chapter 3

Multithreading Thermal Control

3.1 Introduction

In the previous chapter we formulated various bounds for power management policies with the goal of maximizing performance/watt. Through this abstract approach we developed some intuition on the tradeoffs between performance and power for a particular scenario. A drawback of this detached approach, however, is that it does not give any direct sense of how power output gives rise to thermal effects. Because the thermal state of a system is dependent on a number of more complex time-dependent and space-dependent factors, simulating temperature is entirely another level of microarchitectural modeling. In this chapter, and the remainder of this thesis, we examine temperature-aware policies. These techniques for dynamic management remain aware of the tradeoffs between performance and power, but also dynamically respond to temperatures that change with time.

In past studies, a number of adaptive control methods have been proposed for temperature management in uniprocessors. These include global management techniques such as dynamic voltage and frequency scaling (DVFS) and global clock gating, as well as more localized techniques such as fetch/dispatch throttling and register-file throt-

tling [9, 29, 72]. While such techniques have been shown to greatly aid thermal management, recurring challenges involve optimizing the necessary power/performance tradeoffs, ensuring sustained performance, and particularly dealing with hot spots—small sections of a chip attaining temperatures significantly higher than the chip's overall temperature.

When examining thermal issues it is important to explore the problem in the context of prominent architectural paradigms; thus we explore this issue in simultaneous-multithreaded (SMT) processors. SMT is an architectural paradigm that involves issuing instructions such that multiple threads on a single core closely share resources [82]. Various implementations of SMT are now available in several commercial processors [16, 38, 79]. SMT cores seek greater performance by densely packing issue slots and hence can be cause for thermal stress. Our work explores the idea of taking advantage of SMT's added flexibility due to the availability of multiple threads. As a localized technique, we propose that selectively fetching among different programs can allow thermal hot spots to be better controlled and prevented. In our experiment we show that adaptive thread management can tightly control temperature, which has implications for better thermal management and overall reliability [76]. Also, as a localized microarchitectural mechanism, application and design of such adaptive control can work independently or in conjunction with global thermal management techniques such as DVFS.

Our specific contributions are as follows:

- We characterize several benchmarks based on their respective hot spot behaviors. We find that for our processor configuration, each program's hot spot behavior can be characterized largely by its integer register file intensity and floating point register file intensity.

- We propose and evaluate an online adaptive fetch algorithm to take advantage of these heterogeneous characteristics when threads are mixed through SMT. We

find that when operating in the thermally limited region, our algorithm reduces the occurrence of thermal emergencies, increasing performance by an average of 30% and reducing the ED^2 product on the order of 40%. Furthermore, this is a *local* temperature management policy which targets hot spots and can be used in combination, rather than in competition, with global thermal management such as DVFS.

- We repeat these experiments with a similar adaptive algorithm based on selective register naming instead of instruction fetching. For this alternate mechanism, which operates at a later stage in the pipeline, we find correlated but comparatively smaller performance improvements: roughly 70% as effective.

The remainder of this chapter is structured as follows. Section 3.2 presents our simulation infrastructure and methodology. In Section 3.3 we explain our adaptive fetch policy and show our experimental results in terms of measured performance effects and energy savings. In Section 3.4 we perform similar experiments from an adaptive register renaming perspective and compare to the corresponding fetch throttling or adaptive fetch results. Section 3.5 discusses related work, while Section 3.7 summarizes our results.

3.2 Methodology

3.2.1 Simulation Framework

We model a detailed out-of-order CPU resembling a single-core portion of the IBM POWER4™processor with SMT support. Our simulation framework is based on the IBM Turandot simulator [60], which is a predecessor to the simulation framework used in Chapter 2. Dynamic power calculations are provided by PowerTimer, an add-on for Turandot that provides detailed power measurements based on macroblock formations

Global Design Parameters	
Process Technology	0.18μ
Supply Voltage	1.2 V
Clock Rate	1.4 GHz
Organization	single-core
Core Configuration	
SMT Support	2 threads
Dispatch Rate	up to 5 instructions per cycle
Reservation Stations	Mem/Int queues (2x20), FP queue (2x5)
Functional Units	2 FXU, 2 FPU, 2 LSU, 1 BRU
Physical Registers	120 GPR, 90 FPR
Branch Predictor	16K-entry bimodal, 16K-entry gshare, 16K-entry selector
Memory Hierarchy	
L1 Dcache	32 KB, 2-way, 128 byte blocks, 1-cycle latency
L1 Icache	64 KB, 2-way, 128 byte blocks, 1-cycle latency
L2 I/Dcache	2 MB, 4-way LRU, 128 byte blocks, 9-cycle latency
Main Memory	77-cycle latency
Dynamic Control Parameters	
Temperature Sampling Interval	10,000 cycles
Event-counter Sample Window	1,000,000 cycles

Table 3.1: Design parameters for modeled CPU.

derived from low-level RTL power simulations [8]. Integrated with this is the HotSpot 2.0 [35, 74] temperature modeling tool to provide spatial thermal analysis. Unlike the simulator used in Chapter 2, this infrastructure does not require modeling of multiple cores. This model has, however, been extended for SMT support [52].

We model a single-core processor with SMT support on 0.18μ technology. This design level is known to already create significant hot spot effects, a problem which becomes even more prominent at smaller feature sizes. Our design parameters are shown in Table 3.1. Although PowerTimer is directly parameterized based on these options, HotSpot naturally requires additional input to describe the processor's spatial layout. This floorplan is shown in Figure 3.1.

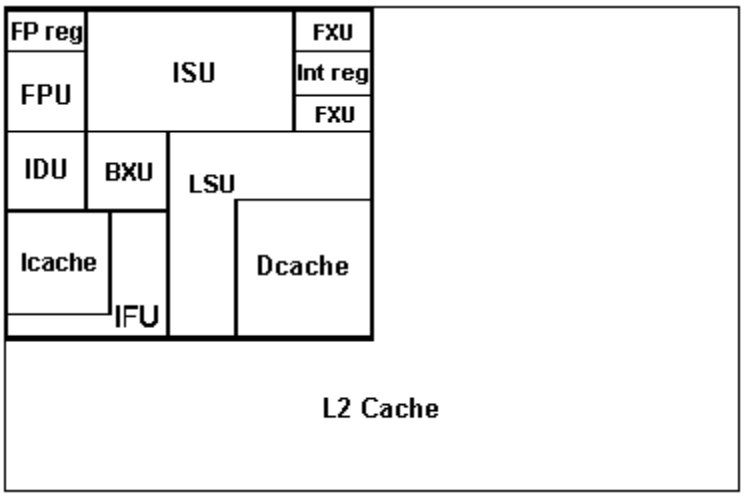


Figure 3.1: Floorplan input to HotSpot 2.0, as also used by Li et al. [51].

Since PowerTimer does not model leakage current by default, an added modification is to model leakage through an area-based empirical equation [34]. Thus, the leakage power of each structure is calculated only by its area and time-dependent temperature. Although more diverse and accurate leakage models do exist [11], this equation is sufficient to model the temperature dependence and quickly derive leakage estimates for all processor structures.

3.2.2 Benchmarks

We analyze workloads based on ten benchmarks obtained from the SPEC 2000 benchmark suite. We have chosen five programs from the integer-based SPECint portion and the other five are from SPECfp, as depicted in Table 3.2.

For outcomes of mixing different programs through simultaneous multithreading it has been shown that the end performance effects can be predicted somewhat based on characteristics of the individual applications [51, 75, 81]. Thus, we also characterize our individual test programs before deciding upon which combinations to mix through multithreading. While hot spots can be unmanageable if their locations vary unpredictably with time, various simulation results [18, 27, 51, 74] have indicated

Name	Benchmark suite	Function	FXU-reg intensive	FPU-reg intensive
188.ammp	SPECfp	computational chemistry	N	Y
173.applu	SPECfp	computational fluid dynamics/physics	N	Y
191.fma3d	SPECfp	mechanical response simulation	Y	Y
178.galgel	SPECfp	computational fluid dynamics	Y	Y
176.gcc	SPECint	C language compiler	Y	N
164.gzip	SPECint	compression	N	N
181.mcf	SPECint	mass transportation scheduling	Y	N
177.mesa	SPECfp	3-D graphics library	Y	N
197.parser	SPECint	word processing	N	N
300.twolf	SPECint	lithography placement and routing	Y	N

Table 3.2: SPEC 2000 benchmarks as selected for this experiment, listed alphabetically.

that for particular processor designs hot spots predictably tend to occur in a handful of locations.

In our design, we find that almost universally the hottest portion of the chip is either the fixed-point execution (FXU) register file or the floating point (FPU) register file. Thus, each of the benchmarks are measured in terms of their thermal intensity for these two chip locations. This measurement is done in advance by executing the programs without thermal control and examining the steady-state and final temperatures on these units. Programs that showed steady-state temperatures above 93°C on units have been marked as such in the two rightmost columns of Table 3.2. For our dynamic policy, later described in Section 3.3.1, it shall be necessary to know the heating characteristics of the running programs. For this, we observe a direct correlation between each program's register file heating characteristics and

Workload	Thermal heterogeneity	Reason
ammp-gzip	significant	Floating point benchmark mixed with an integer benchmark.
ammp-mcf	significant	Floating point benchmark mixed with an integer benchmark.
applu-parser	moderate	Can exploit parser's extremely low IPC to cool either hot spot.
applu-twolf	significant	Floating point benchmark mixed with an integer benchmark.
fma3d-galgel	small	Both benchmarks are high-intensity on both register files.
fma3d-twolf	small	Both benchmarks are integer-intensive.
galgel-mesa	moderate	Both benchmarks are integer-intensive, but mesa is greater.
gcc-mesa	small	Two integer benchmarks.
gcc-parser	moderate	Two integer benchmarks, but parser's slowness needs management.
gzip-mcf	small	Two integer benchmarks.

Table 3.3: Multithreaded benchmark mixes. Pairs with a higher degree of thermal heterogeneity show greater promise in benefitting from SMT-specific adaptive thermal management.

the number of register file accesses recorded, and we are able to universally use this observed ratio in our dynamic policy when applied to any workload.

We then use this data in deciding how to appropriately create a set of ten SMT-based workloads. First, we would like to mix programs which show opposite thermal behaviors since these give the greatest potential for adaptive thermal control. These include mixing integer intensive programs with floating-point intensive programs. For the other end of the spectrum, we also include several test cases which lack thermal heterogeneity, such as pairs of floating point benchmarks and pairs or integer-only benchmarks. In such scenarios we might not expect a significant benefit from thread-sensitive thermal control, but it is important to show that our algorithm can at least be ensured not to be detrimental in these cases. A list of these chosen workloads and their corresponding qualitative characterizations can be found in Table 3.3.

In order to simulate only representative portions of these programs, we use SimPoint [64, 70] with sampling intervals of 100 million instructions in order to obtain all relevant traces executed in our experiments. To simulate relevant temperature behavior on such a short time interval, we choose an operating point and thermal threshold such that thermal triggers come into play approximately 60% of the time for most our test workloads. This may also be described as a "duty cycle" [66] of about 40%, although our throttling techniques do not cater strictly to the duty cycle definition: During throttled cycles, execution may still continue outside of the fetch and rename stages. (Later on, in Chapter 4, we explore core-wide throttling techniques where the strict definition of duty cycle becomes more applicable.) As shown by our data later in this chapter, the time spent in thermal emergency mode naturally decreases when applying our adaptive policies.

3.2.3 Metrics

As one measure of the performance impact of our technique, we use the criterion of *weighted speedup* as described by Snavely and Tullsen [75] shown below.

$$Weighted\ Speedup = \sum \frac{IPC_{SMT}[i]}{IPC_{normal}[i]}$$

This is intended to be a fair comparison between two executions and prevents biasing the metric on policies that execute unusual portions of high-ILP or low-ILP threads. Note that the $IPC_{SMT}[i]$ is only a portion of the multithreaded system's total IPC.

In these experiments the $IPC_{normal}[i]$ denominator is measured under thermally limited conditions. To be specific, all executions start with temperature profiles where both register files are just barely below the thermal threshold of 85°C. Under these conditions we find that a number of workloads are thermally throttled about 60% of

the time, and this affects the denominator in the above equation. This method is largely different from other works [18, 75, 81], where weighted speedup is measured assuming the baseline single-threaded executions are not thermally constrained in any way.

For the purpose of this study, the IPC measurements are all done in thermally constrained mode. This is appropriate since we seek to analyze behavior particularly in the thermally limited region. Weighted speedup is meant to qualify as a fair raw performance metric, similar to how IPC alone is sometimes used in uniprocessor comparisons. While the weighted metric is arguably more qualified for our purposes (examining performance improvement in SMT processors), for most of our results the overall workload IPC is also strongly correlated to weighted speedup anyway.

Even with both the IPC and weighted speedup metrics showing positive results, however, thread prioritizing policies can generate situations that are "unfair" to some threads. To quantify this drawback, we also directly present the ratios of thread retirement on a per-thread basis for each of our tested workloads.

In order to appropriately measure performance from a power-aware perspective, for our second main metric we use the established energy·delay-squared product (ED^2). This now widely used metric realistically takes into account tradeoffs between power and energy in the context of DVFS, where scaling the voltage can have a cubic effect on power reduction. Since our proposed adaptive policy is a local mechanism, it can still be combined with global power reduction techniques such as DVFS, thus making ED^2 a relevant metric for comparison. Since we are measuring workloads that complete with different instruction counts and instruction mix ratios on different parameterizations, we must normalize the ED^2 metric to a *per instruction* basis. We use the following formula to calculate this metric from the IPC and energy per instruction, EPI.

$$ED^2 = \frac{EPI}{IPC^2 * clock\ frequency^2}$$

We use IPC, per-thread throughput, and ED^2 as primary means to examine the performance of our policies. However, there are certainly other possible metrics. For example, for interactive applications, it would be interesting to see how the user-perceived latency is affected by our policies. Although the work in this chapter does not delve into all possible delay metrics, we believe that the measurements on core instruction throughput and thread fairness provide a reasonable proxy for how other metrics would be expected to scale.

3.3 Adaptive Thermal Control

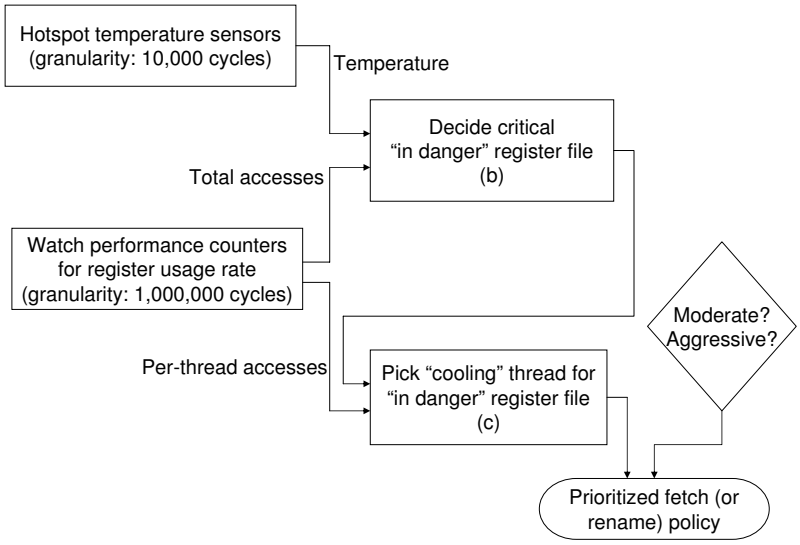
3.3.1 Adaptive Control Algorithm Overview

Our adaptive control is based on the input of temperature sensors that exist in many modern commercial processors. Although the exact placement of the POWER5™processor's 24 available sensors is unknown [16], it is reasonable to assume that at least two of these would be allocated to the register file locations which are primary potential hot spots. We use 85° C as the threshold temperature for enacting thermal control. As modern commercial microprocessors tend to list maximum allowable operating temperatures in the range of 70 to 90° C [17] we feel this is a reasonable choice. Utilizing the dynamically profiled thread behavior information, our decision algorithm for adaptive thread selection is implemented on top of this as follows:

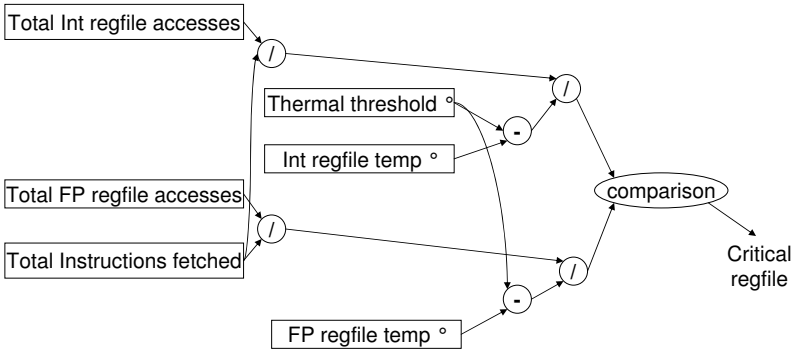
For the actual adaptive technique of dynamic thermal management, we modify the default round-robin SMT fetch policy originally implemented for Turandot in [52]. Our modifications target thermal control logically by avoiding integer-intensive benchmarks when the FXU register file's temperature appears more likely to reach the temperature threshold, and likewise to reduce the execution rate of floating point intensive benchmarks when the FPU register file goes above its threshold. In order for the processor to identify whether running programs are integer-intensive or floating

point intensive, we need some kind of profiling mechanism to determine the properties of running programs. One method for this is to dynamically sample hardware event counters. As mentioned in Section 3.2.2, our goal, using this profiled information, is to exploit a direct correlation between register file accesses and the long-term steady state register file temperature. Powell et al. also use counter information as such to predict heating behavior for key resources [66], and recent work by Lee and Skadron has shown that hardware performance counters can be reliably used to predict temperature effects on real systems [46].

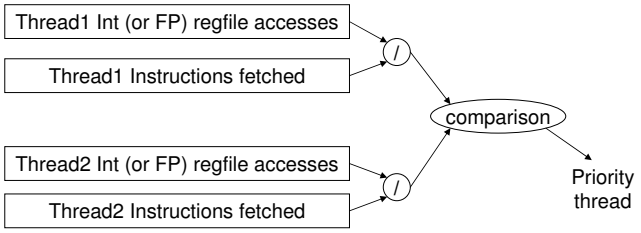
Figure 3.2 (a) shows a high-level view of the decision policies for thermally adaptive multithreading. When the processor is not in thermal arrest mode, the difference between the thermal threshold and the integer register file's temperature is calculated. At the same time, from profiling we obtain the average number of integer register file accesses per fetched instruction for each of the two threads. Using a calibrated threshold—in terms of PowerTimer's internal access counters—we decide whether the integer register file is in danger of approaching our specified maximum temperature (85°). We also do all of the above for the floating point register file and compare to see which unit is potentially in danger. The steps necessary to calculate and decide this are depicted graphically in Figure 3.2 (b). Our adaptive policy then takes effect. Its goal is to choose instructions from the thread that are either likely to cool or less quickly heat the hotter of the two register files. Once the potentially hotter of the two units is identified, the decision as to which thread to pick from is decided by choosing the thread measured to be less intensive on the integer register file—or floating point register file, if applicable—based on the thread's dynamically profiled measure of register file accesses per issued instruction. This second stage of the decision process is depicted in Figure 3.2 (c). These calculations are done only once per temperature measurement cycle, and thus may be precalculated with delay in such a way that it does not affect the fetch logic's critical path.



(a) High-level view of the decision process for adaptive SMT.



(b) Portion of our algorithm that determines which unit is in thermal danger.



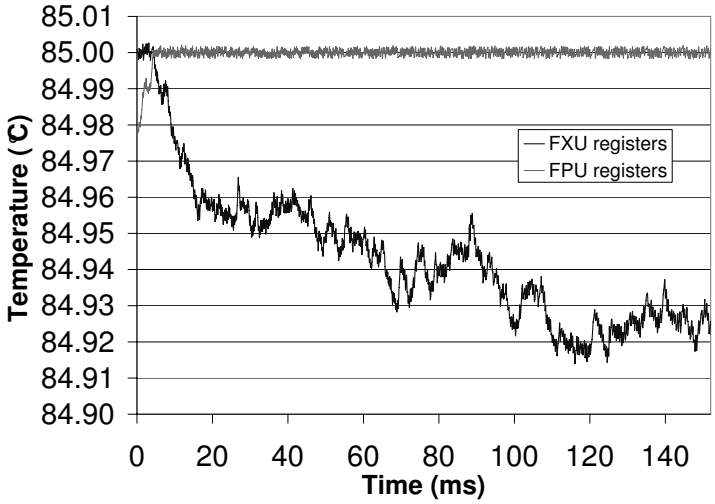
(c) Decision algorithm for which thread is selected if the integer (or floating-point) register file is judged to be in danger.

Figure 3.2: Flow chart depicting an overview of the adaptive SMT algorithm, and two block diagrams demonstrating the calculation and decisions, which also reflect the added components used in a hardware design.

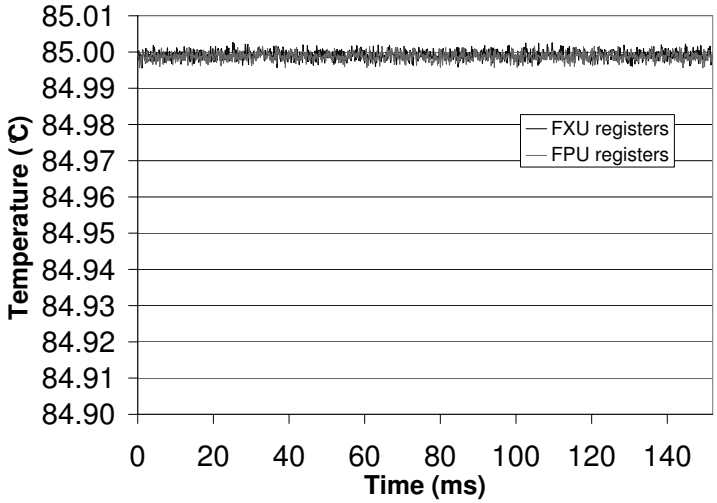
Fetch priority adjustment is in many ways an extension of basic fetch throttling on a uniprocessor. Also known as toggling, throttling involves simply disabling instruction fetch whenever a section of the processor surpasses the specified thermal threshold [9]. Once this mechanism has been triggered, ideally the processor would quickly cool down until it goes below the thermal threshold and can continue normal operation. Fetch throttling thus forms the comparison baseline for our measurements. In actuality, our fetch priority adjustment system is not an alternative but rather runs in combination with fetch throttling. Since thermal stability cannot be ensured if instructions are always issued—as is the case when all available threads are thermally intensive—it is necessary for our design to have a backup policy to fall back on in order to guarantee prevention of thermal violations. Figure 3.3 shows a sample of the effects of thermal management under our proposed algorithm from a time-dependent perspective. In the baseline fetch throttling example of Figure 3.3 (a) one hot spot can remain the primary performance hindrance, while with our adaptive algorithm in Figure 3.3 (b) instructions from each thread can be issued such that the two key hot spot temperatures remain close.

3.3.2 Adaptive Control Algorithm: Other Issues and Discussion

To avoid unpredictable cases of thread starvation, we allocate a portion of cycles where the default fetch policy holds regardless. For our policy labeled "moderate", the first two cycles out of every four cycles default to the standard alternation among threads (round-robin) policy. This ensures a degree of thread fairness fairly close to the original policy, but at the potential expense of poorer thermal management. Our "aggressive" policy allocates only the first two out of every sixteen cycles for defaulting to the round-robin policy, pushing a stronger tradeoff between thread fairness and thermal management. While our current fallback policy is round-robin, for future



(a) Baseline fetch throttling thermal control.



(b) Temperature-aware thread fetch policy.

Figure 3.3: Transient hot spot temperatures for fma3d-twolf workload under our baseline and adaptive policy. Although only representative of a short time interval, the control policy shown in (b) is capable of tuning temperature on a very fine-grain level.

work we hope to extend our framework to use more real world-applicable fetch policies including ICOUNT [82]. Such designs, which were originally aimed for aggressive performance, may become more severely penalized under thermally limited conditions and hence would likely benefit more from our temperature-aware policies.

For identifying the heat behavior of each thread, we sample its execution through

performance counters at runtime. Our current dynamic profiling utilizes event counts starting back 100 temperature measurement cycles (1,000,000 CPU cycles) through the point in time of the most recent temperature sample. Being two orders of magnitude larger than the temperature measurement cycle, we ensure that profiling hardware interferes very little with the main pipeline.

We do not model sensor error, although sensor delay is modeled: temperature is recalculated only every 10,000 cycles. At the given clock rate this amounts to about 6 μs of sensor delay. Thus, any hardware necessary for recalculating the temperature and feeding it to the control logic cannot be expected to affect the critical path of the pipeline, as the result is precalculated and fed in with appropriate delay. We find under this model that it usually takes between one and three measurement cycles (10,000 to 30,000 CPU cycles) to fall back below the thermal threshold after each thermal threshold breach is detected. Despite thermal emergencies occurring often throughout execution, there is no additional delay penalty for enacting thermal control. Compared to DVFS, this is a key advantage of pure microarchitectural techniques, as demonstrated by Brooks et al. [9].

Heo et al. [34] have shown that designs enacting thermal control on a sufficiently fine-grain interval pose an advantage for tightly controlling temperatures, although they can be more costly in terms of other design factors. However, it seems feasible that this mechanism could be moved to the operating system level, as Powell et al. have demonstrated that thermal fluctuations happen on a sufficiently coarse grain time interval adequate to be managed by the OS [66]. Hybrid techniques involving both the microarchitecture and OS are also a possible implementation. To be specific, prioritized fetching and renaming can be performed by the microarchitecture, while numerical specifications of those process priorities can be dictated by the OS depending on thermal conditions. In Chapter 4, we explore some related hybrid approaches.

For implementing the control algorithm in real hardware, event counters are necessary to measure (in total as well as on a per-thread basis) integer register file accesses per cycle, floating point accesses per cycle, and number of instructions fetched per cycle. In addition, calculation hardware is needed including adders, a division unit, and necessary decision logic. Note that although our algorithm as depicted in Figure 3.2 shows as many as eight dividers, in reality only a single shared divider is necessary since speed of calculation is not critical. Since these calculations would be invoked only once for every temperature measurement cycle, the energy overhead is negligible. For perspective, modern DVFS solutions employ PID-based hardware which involves even more additional gates but also has insignificant energy overhead while not affecting the microprocessor pipeline's critical path.

3.3.3 Experimental Results

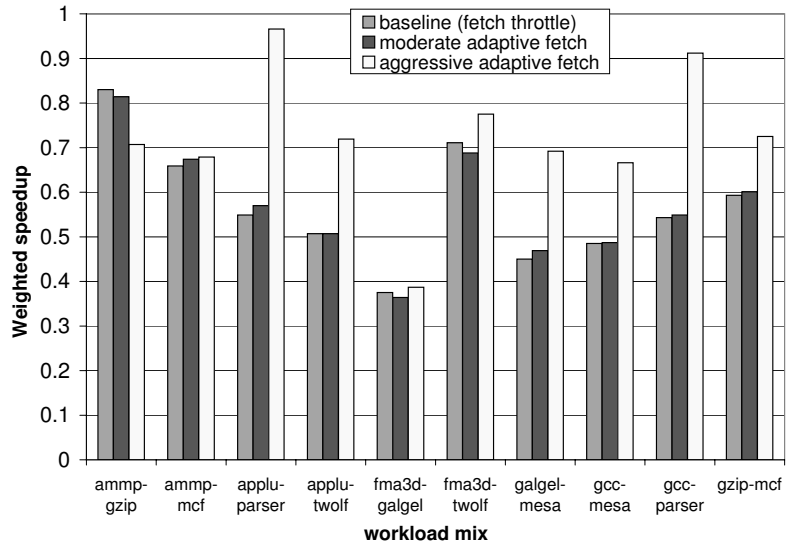
In this section we examine the benefits of temperature-aware adaptive thread priority management. Table 3.4 lists performance and power metrics for all mixes under the baseline control method. The weighted speedup for each of these mixes is small, notably less than 1.0 in all cases. This signifies a cost associated with simultaneous multithreading, and it is primarily due to operating in the thermally limited region. It is this cost we seek to address. Although all mixes have weighted speedups greater than 1.0 when operating below the thermally limited region, the increased temperatures due to high instruction issue rates, however, make SMT actually detrimental to performance in this region. For these executions the bottleneck hot spots are still the integer and floating point register files where one or the other hovers at the thermal threshold of 85°C. The overall chip temperature as reflected by its large L2 cache remains at approximately 52°C, more than 30° less. The thread retire ratio gives an overview of the fairness in each pair of programs. The first percentage represents the fraction of retired instructions that belong to the first program, and the second

mix	IPC	thread retire ratio	weighted speedup	ED^2 ($\frac{J \cdot s^2}{instr^3}$)	throttle rate
ammp-gzip	0.820	57.8%/42.2%	0.830	2.83e-26	44.9%
ammp-mcf	0.410	66.7%/33.3%	0.659	2.23e-25	49.2%
applu-parser	0.527	63.5%/36.5%	0.549	9.87e-26	65.1%
applu-twolf	0.486	60.6%/39.4%	0.507	1.27e-25	65.2%
fma3d-galgel	0.583	43.4%/56.6%	0.375	7.48e-26	69.1%
fma3d-twolf	0.716	60.4%/39.6%	0.711	4.17e-26	51.9%
galgel-mesa	0.708	60.8%/39.2%	0.450	4.27e-26	66.8%
gcc-mesa	0.494	53.0%/47.0%	0.485	1.22e-25	67.6%
gcc-parser	0.485	62.0%/38.0%	0.543	1.30e-25	64.9%
gzip-mcf	0.304	57.1%/42.9%	0.593	5.25e-25	55.4%

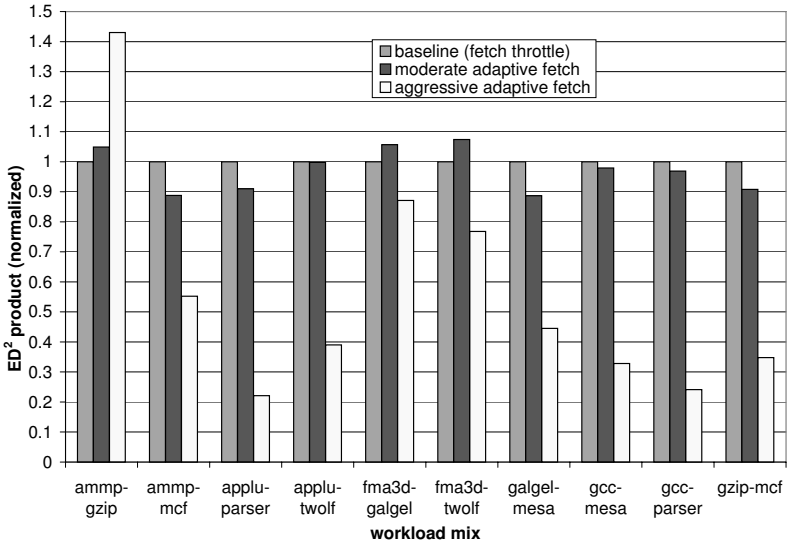
Table 3.4: Baseline results for fetch toggling based DTM without adaptive thread control.

percentage represents the corresponding fraction for the second program.

Improving on this baseline, Table 3.5 lists performance and power metrics for all mixes under adaptive thread fetching for our moderate and aggressive-level policies. Figure 3.4 pulls together the primary parameters as presented in Tables 3.4 and 3.5 and presents these results graphically. Note that in most cases where heterogeneously behaved programs are mixed, we see a 30-40% IPC improvement with a similar increase in weighted speedup. This performance improvement is directly caused by a corresponding reduction in the amount of time the processor is throttled. This can be directly seen by the throttle ratio given in the three tables. Next, the ED^2 reduction is related to IPC parabolically and can be explained as follows. Dynamic power increases proportionally with higher IPC, but this does not significantly reduce EPI since the amount of work performed per instruction remains the same. Leakage power, on the other hand, remains mostly unchanged since our overall chip temperature remains largely unaffected, resulting in somewhat lower energy *per instruction* as leakage in this model constitutes only about 25% of total power. Thus the key factor causing a parabolically correlated decrease in ED^2 reduction is the delay term squared.



(a) Weighted speedup for all workloads under three thermal-aware fetch policies.



(b) Corresponding normalized ED^2 product for these workloads.

Figure 3.4: Weighted speedup and ED^2 for fetch-based dynamic thermal management.

As expected, we find that our adaptive fetch technique offers the biggest improvement in cases allowing a high degree of thermal variety in workload mixes. For other cases such as gzip-mcf and gcc-mesa (integer only), we see there is actually a significant performance potential despite the constituent programs being similar in terms of register file usage. The exploitable difference here is perhaps that although nei-

(a) Moderate adaptive fetch management.

mix	IPC	thread retire ratio	weighted speedup	ED^2 ($\frac{J \cdot s^2}{instr^3}$)	throttle rate
ammp-gzip	0.806	58.9%/41.1%	0.814	2.97e-26	45.2%
ammp-mcf	0.427	68.1%/31.9%	0.674	1.98e-25	48.0%
applu-parser	0.545	61.8%/38.2%	0.570	8.99e-26	64.8%
applu-twolf	0.486	61.0%/39.0%	0.507	1.27e-25	65.3%
fma3d-galgel	0.572	41.4%/58.6%	0.364	7.90e-26	69.7%
fma3d-twolf	0.698	62.6%/37.4%	0.688	4.47e-26	49.1%
galgel-mesa	0.738	60.7%/39.3%	0.469	3.79e-26	64.9%
gcc-mesa	0.498	50.5%/49.5%	0.487	1.19e-25	67.1%
gcc-parser	0.490	60.4%/39.6%	0.549	1.25e-25	65.1%
gzip-mcf	0.314	59.0%/41.0%	0.601	4.77e-25	55.0%

(b) Agressive adaptive fetch management.

mix	IPC	thread retire ratio	weighted speedup	ED^2 ($\frac{J \cdot s^2}{instr^3}$)	throttle rate
ammp-gzip	0.720	70.3%/29.7%	0.707	4.05e-26	38.4%
ammp-mcf	0.498	78.4%/21.6%	0.679	1.23e-25	41.0%
applu-parser	0.892	47.5%/52.5%	0.966	2.19e-26	17.4%
applu-twolf	0.675	51.1%/48.9%	0.719	4.96e-26	63.4%
fma3d-galgel	0.611	40.8%/59.2%	0.387	6.52e-26	60.8%
fma3d-twolf	0.784	62.1%/37.9%	0.775	3.20e-26	48.2%
galgel-mesa	0.936	35.6%/64.4%	0.692	1.90e-26	31.3%
gcc-mesa	0.719	22.5%/77.5%	0.666	3.99e-26	39.9%
gcc-parser	0.787	32.8%/67.2%	0.912	3.12e-26	10.7%
gzip-mcf	0.436	72.3%/27.7%	0.725	1.83e-25	47.2%

Table 3.5: Complete data for workload behavior under our adaptive thread fetching policy.

ther program uses floating point operations, these programs already possess much imbalance in terms of their frequency of integer accesses. One workload, ammp-gzip, shows a decrease in performance under our algorithm. This at first seems surprising since it is a heterogeneous workload—containing one integer and one floating-point benchmark—that should have potential for balancing. The cause is that the baseline case using throttling happens to be already very balanced with starting and ending temperatures for each register file remaining close to each other. This most likely happens by chance; if larger or different execution traces for the two programs are

selected, the temperatures could easily imbalance without adaptive thread management.

The potential cost of our adaptive policy is reduced thread execution fairness as compared to the basic round-robin policy. Overall, we find that the moderate adaptive policy performs better than the baseline with an average of only 1% improvement in terms of weighted speedup or IPC. Our aggressive policy performs significantly better than the moderate policy showing an average of 30% improvement in terms of weighted speedup. The ED^2 product, largely correlated, averages 44% reduction under the aggressive adaptive policy.

3.4 Adaptive Register Renaming

3.4.1 Design Description

Our second set of experiments is much like the first, except it involves adaptive control at a later stage of the pipeline, namely the register renaming logic. Our adaptive rename policy is exactly the same as explained earlier for adaptive fetch control, except instead of being fetch-based it controls the priority at which a thread receives the register renaming service. For deciding which thread to give renaming priority to on each cycle, we use the same decision policy as depicted in Figure 3.2. When the decision to rename registers for only a particular thread is decided on any given cycle, the register rename hardware maps registers only for the selected thread, effectively stalling services for the other thread. Likewise, instead of fetch throttling serving as our baseline thermal control method, we compare against basic rename throttling [51] instead. This involves simply disabling the rename logic when the processor appears above its thermal threshold.

A difference, and possible benefit from the selective renaming technique, is that it operates closer to the hot spot of interest, namely the register file. From Figure

3.2 it can be seen that much of the profiling logic is required to gather information from the register file hardware. In the case of adaptive fetching, this would require communication between the register files and the fetch stage of the pipeline. An adaptive renaming scheme, however, may require shorter and less intrusive global wires as its operation is restricted to the register file stage of the pipeline.

One clear drawback, however, is that throttling at a later stage of the pipeline allows instructions to enter the pipeline and consume resources. These extra instructions, still in the early stage of the pipeline, may still consume power even as their operations cannot progress further in the pipeline while being throttled at the register renaming stage. Furthermore, if the instruction window is filled with instructions that remain there, this can limit the capacity for the non-throttled thread to continue fetching. Whether or not these factors are significant may depend largely on how power-consuming the pipeline front-end is, and whether the instruction window is capacity-limited most of the time.

3.4.2 Experimental Results

Our baseline results regarding rename throttling without adaptive register renaming are shown in Table 3.6. We find the efficacy of this alternative thermal management technique to be on the same order of efficacy as fetch throttling, a result consistent with recent work by Li et al. [51].

We enact the adaptive register renaming strategy described in Section 3.4.1. As with our other fetch-based experiments, note that this is not an alternative to basic register rename throttling but rather is operating on top of the parent policy so as to ensure thermal stability. Table 3.7 shows all corresponding data for the adaptive renaming experiments, and likewise for comparison Figure 3.5 brings together the main results of Tables 3.6 and 3.7 to compare graphically. The pattern of measurable performance improvement in terms of ED^2 is much the same as is found from our

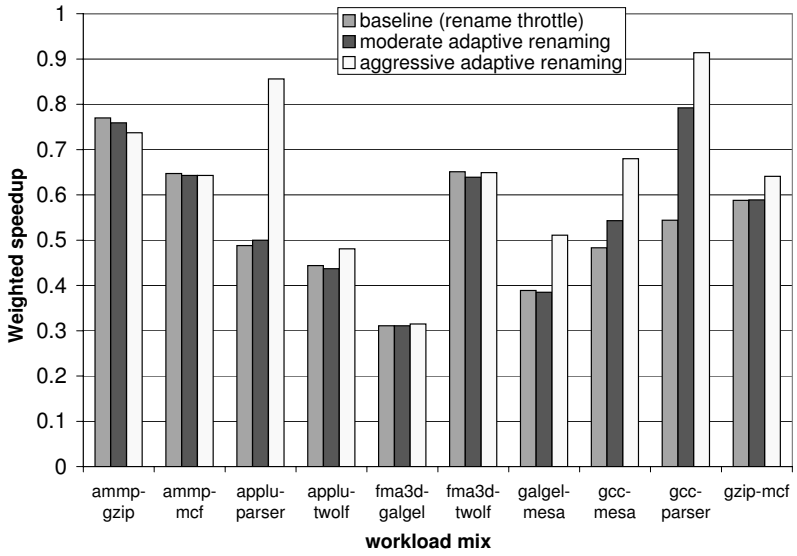
mix	IPC	thread retire ratio	weighted speedup	ED^2 ($\frac{J \cdot s^2}{instr^3}$)	throttle rate
ammp-gzip	0.760	57.7%/42.3%	0.770	3.56e-26	48.6%
ammp-mcf	0.402	66.6%/33.4%	0.647	2.41e-25	49.6%
applu-parser	0.467	63.2%/36.8%	0.488	1.44e-25	69.2%
applu-twolf	0.425	60.3%/39.7%	0.444	1.92e-25	69.6%
fma3d-galgel	0.484	43.1%/56.9%	0.311	1.32e-25	74.8%
fma3d-twolf	0.655	60.3%/39.7%	0.651	5.46e-26	53.4%
galgel-mesa	0.612	60.8%/39.2%	0.389	6.64e-26	71.2%
gcc-mesa	0.492	53.0%/47.0%	0.483	1.28e-25	67.8%
gcc-parser	0.486	61.8%/38.2%	0.544	1.32e-25	65.1%
gzip-mcf	0.302	57.1%/42.9%	0.588	5.49e-25	55.7%

Table 3.6: Baseline results for rename-throttling based DTM without adaptive thread-specific renaming.

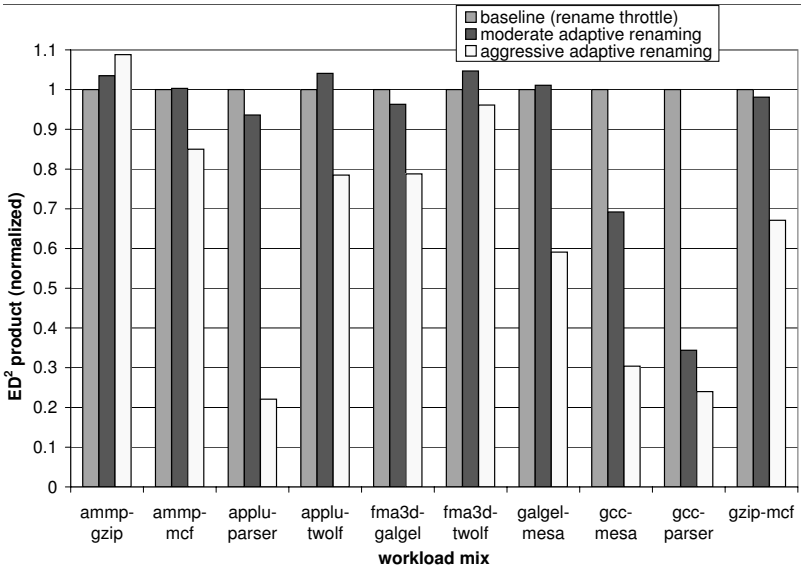
fetch-based experiments. That is, we see roughly the same pattern of performance gains in certain workloads. As mentioned earlier, a drawback expected from throttling at the rename stage is that the register renamer is a later stage of the pipeline, thus unlike fetch management it gives more potential for unwanted instructions to enter the pipeline and consume resources while throttled. Despite this possible downside, the potential for thermal control at this pipeline stage in addition to the fetch stage appears quite viable.

3.5 Related Work

A number of works have examined the power and energy properties of SMT without regard to spatial temperature analysis [42, 52, 67, 68]. Many of these studies tend to explore SMT in comparison to multicore processors. Several other works extend beyond these and examine the thermal properties of SMT [18, 51, 66]. In addition to characterizing SMT's thermal behavior, a number of thermal management techniques for SMT processors have been proposed and studied. For instance, Li et al. [51] experiment with dynamic voltage scaling and localized throttling techniques. How-



(a) Weighted speedup for all workloads under the three thermal-aware renaming policies.



(b) Corresponding ED^2 product for these workloads, normalized.

Figure 3.5: Weighted speedup and ED^2 for register renaming-based dynamic thermal management.

ever, all their tested techniques are applicable to superscalar processors and other paradigms as well; hence, they do not explore SMT-specific constructions. Powell et al. [66] explore SMT thermal management in the context of hybrid SMT-CMP systems and they propose scheduling schemes for optimal scheduling on thermally

(a) Moderate adaptive register renaming.

mix	IPC	thread retire ratio	weighted speedup	ED^2 ($\frac{J \cdot s^2}{instr^3}$)	throttle rate
ammp-gzip	0.751	58.4%/41.6%	0.759	3.69e-26	50.0%
ammp-mcf	0.401	67.0%/33.0%	0.643	2.42e-25	49.5%
applu-parser	0.477	62.7%/37.3%	0.500	1.35e-25	68.1%
applu-twolf	0.419	61.3%/38.7%	0.437	2.00e-25	69.6%
fma3d-galgel	0.491	40.5%/59.5%	0.311	1.27e-25	74.5%
fma3d-twolf	0.645	61.3%/38.7%	0.639	5.71e-26	53.6%
galgel-mesa	0.610	61.8%/38.2%	0.385	6.71e-26	70.9%
gcc-mesa	0.558	47.7%/52.3%	0.543	8.83e-26	63.9%
gcc-parser	0.703	56.8%/43.2%	0.792	4.55e-26	53.3%
gzip-mcf	0.304	57.5%/42.5%	0.589	5.38e-25	55.7%

(b) Aggressive adaptive register renaming.

mix	IPC	thread retire ratio	weighted speedup	ED^2 ($\frac{J \cdot s^2}{instr^3}$)	throttle rate
ammp-gzip	0.736	62.5%/37.5%	0.737	3.88e-26	40.5%
ammp-mcf	0.423	70.9%/29.1%	0.643	2.05e-25	48.1%
applu-parser	0.786	46.1%/53.9%	0.856	3.19e-26	33.4%
applu-twolf	0.461	61.7%/38.3%	0.481	1.51e-25	69.5%
fma3d-galgel	0.526	31.9%/68.1%	0.315	1.04e-25	68.4%
fma3d-twolf	0.663	64.9%/35.1%	0.649	5.24e-26	53.3%
galgel-mesa	0.733	45.9%/54.1%	0.511	3.92e-26	48.7%
gcc-mesa	0.733	23.4%/76.6%	0.680	3.88e-26	37.7%
gcc-parser	0.788	32.7%/67.3%	0.914	3.18e-26	8.7%
gzip-mcf	0.346	62.1%/37.9%	0.641	3.68e-25	52.2%

Table 3.7: Complete data for workload behavior under our adaptive register renaming policy.

constrained designs. However, their design intervenes only through the operating system and they do not explore more fine-grain techniques that could enable direct thermal management without requiring context switches.

El-Assawy et al. propose SMT-specific extensions targeting another reality of physics for modern processors—the inductive noise problem [24]. Similar to our reasoning, they see SMT providing an opportunity to exploit program diversity in order to counteract with adaptive control.

Hasan et al. [32] propose a mechanism that strongly relates to the design in

this chapter. They envision a scenario whereby a malicious thread may cause a microarchitectural Denial of Service (DoS) attack, and propose remedies for detecting and mitigating the effects of such attacks. However, they do not examine how to optimize SMT operation in terms of naturally occurring thermal stress. Our work here explores this as a general problem to be addressed as processor designs are bound to become more thermally stressed in the future and operate under thermally constrained conditions. Our proposed framework manages to generalize protective thermal arbitration to all programs, including programs that could potentially be intended for malicious attacks.

Since the initial publication of our work [21], the concepts of SMT-specific thermal control have even been demonstrated on actual hardware. In a hybrid approach, Kumar et al. demonstrate similar multithreaded thermal control in combination with clock throttling on an actual Pentium 4 machine [43].

3.6 Future Extensions

For one possible future direction, it would be interesting to see if this method can be extended beyond 2-context SMT to readily scale to greater numbers of threads. While the logic for comparing two threads based on a critical resource's temperature could be extended to manage more than two threads, it is not clear how to fairly and practically partition many threads in terms of allowed execution share. Similarly, although the processor modeled in this chapter is an example of only a single-core single-socket system, another open question would be how these policies may coordinate and provide performance benefits in multicore multiple-socket systems where each core features SMT. Another possibility for extending this work involves testing how it may be implemented alongside complex fetch policies such as ICOUNT.

3.7 Summary

This study proposes and tests a novel form of adaptive DTM specific to SMT processors. At adaptive thread fetching can predictably control temperature of hot spots at a fine grain level. We have found thread priority management provides a weighted speedup performance increase over our conventional fetch toggling technique by an average of 30%, and ED^2 reductions averaging 44% for our test cases. Our analogous experiments dealing with adaptive renaming found strikingly similar results averaging 23% weighted speedup improvement and 35% ED^2 reduction.

Our work demonstrates a heuristic algorithm for a simple case of two primary hot spots on an SMT processor. Future process technologies bring greater thermal challenges including wider gaps between overall chip temperature and localized hotspots. We expect this to worsen and create increased demand for smart thermal control applicable to varied workloads. Such systems pose a challenge but a wider variety of hot spots also brings potential for more advanced adaptive control methods.

Our proposed algorithm makes a clear tradeoff between baseline thread fairness and sustaining performance. It is most applicable in systems which allow a wide degree of thread priority and scheduling freedom. This would include systems such as scientific computing environments where many huge workloads are queued up without strict process priorities. One can also envision, for example, a thermally constrained server system where one might find it more appropriate to fairly allocate user time based on its thermal cost (power) rather than direct CPU-cycle cost. A mechanism such as this one directly enables such an energy-guided quota. Instead of requiring overly specific protections against malicious thermal attacks [32], our technique serves as both a safeguard and a standard policy for resource sharing in a thermally constrained environment.

Chapter 4

Multicore Thermal Control

4.1 Introduction

The previous chapter demonstrates an adaptive technique using the flexibility of multiple applications on a single-core processor. As we have emphasized, however, multicore processors are becoming the dominant architectural paradigm in modern times. While simultaneous multithreading provides flexibility in adaptive policies based on available applications, multicore architectures additionally provide spatial flexibility in the placement of those applications. In this chapter we organize the design space for multicore thermal control and explore a range of thermal control possibilities.

Although Chapter 2 also examined multicore processors, it did not examine these platforms in the context of thermal management. This chapter looks at thermal properties of applications running on multicore processors and the corresponding thermal management techniques. Unlike Chapter 2, however, here we do not experiment with parallel applications, and instead focus primarily on multiprogrammed workloads. One advantage of the multiprogramming scenario is that it manages to simplify our analysis and policies. We implement feedback-control and migration, which can be

successful because applications can execute largely independent of one another. Second, many parallel applications, such as those examined in Chapter 2 exhibit fairly uniform behavior among multiple threads, rather than creating chaotic power and temperature variations. Workloads formed from several different applications, on the other hand, represent a sufficient challenge for which adaptive management is most applicable.

We begin our analysis by dividing the CMP thermal design space into a taxonomy of orthogonal design choices. This taxonomy allows us to systematically and quantitatively explore the thermal design space. In some parts of the space, we quantify the benefits of useful combinations of previously-proposed approaches. In other parts of the space, however, we propose novel thermal control techniques and quantify their value. For example, one of the key novelties of the chapter lies in our use of formal control theory techniques to propose, design, and evaluate a multi-loop control mechanism that allows the operating system and the processor hardware to collaborate on a robust, stable, and effective thermal management policy. On the initial publication of our work [22], our proposal was the first thermal management technique to exploit multi-loop control.

The contributions of this chapter are as follows:

- Distributed DVFS provides considerable performance improvement under thermal duress, on average improving throughput by 2.5X relative to our baseline. While the design complexity cost of multiple clock domains is considerable, we show that the performance potential is significant as well.
- When independent per-core DVFS controls are unavailable, we find that other options perform well. In particular, a thread migration policy without per-core DVFS can still improve performance by as much as 2X.
- These methods can be combined through our sensor-based migration policy involving multi-loop control. The operating system engages in coarse-grained

control and migration, while the hardware level engages in a finer-grained level of formal control based on DVFS. We find that this method offers up to 2.6X improvements over baseline.

Overall, this chapter offers insights on the combined leverage of DTM methods, on the value of distributed DVFS for thermal management, and on the robustness and feasibility of formal multi-loop control via OS-processor collaborations. Given the importance of thermal design in current and future processors, these contributions represent useful next steps for the field of thermal-aware architecture.

The remainder of this chapter is structured as follows. Section 4.2 presents our taxonomy for thermal control. Section 4.3 describes our simulation environment and experiment methodology to quantify the properties of these systems. Section 4.4 examines implementation issues for our control methods. Sections 4.5 through 4.7 show our experimental results. Section 4.8 discusses related work, and Section 4.9 summarizes this chapter.

4.2 Thermal Control Taxonomy

Since temperature is largely dependent on power output over time, general power-reduction techniques are typically good first steps for temperature-aware design. In addition to aggregate heat production, however, there can be significant temperature variance across different regions of the die, and thus one also must worry about more localized hot spots at particular portions of the chip.

4.2.1 Thermal Control Taxonomy

For controlling hot spots, one can either (a) *reduce* heat production, or (b) *balance* heat production. Under this reasoning, our work seeks to classify DTM schemes in a systematic manner so that we can characterize and quantify their design tradeoffs

	No migration		Counter-based migration		Sensor-based migration	
	Stop-go	**DVFS**	**Stop-go**	**DVFS**	**Stop-go**	**DVFS**
Global	Stop-go	Global DVFS	Stop-go + counter-based migration	Global DVFS + counter-based migration	Stop-go + sensor-based migration	Global DVFS + sensor-based migration
Distributed	Dist. stop-go	Dist. DVFS	Dist. stop-go + counter-based migration	Dist. DVFS + counter-based migration	Dist. stop-go + sensor-based migration	Dist. DVFS + sensor-based migration

Table 4.1: Thermal control taxonomy, forming twelve possible thermal management schemes.

taken both individually and in combinations.

Our taxonomy is depicted in Table 4.1. We regard our policy decisions as a set of orthogonal axes. One axis refers to the type of low-level control employed. Among the choices we explore are that of a stop-go policy (turn off a core or the whole chip when thermal management indicates the temperature should be reduced) and DVFS (apply voltage-frequency scaling to reduce temperature). Our second axis is that of deciding whether to use a global controller for all cores, or whether to use distributed per-core approaches. Per-core decisions may require more complex hardware, but in turn will let the system respond more individually to the needs of different applications running on each core. Our final axis regards a process migration policy, which acts on a more coarse-grain time scale. Options here are to never migrate threads (the base case) or to migrate threads in response to either thermal-sensor readings, or counter-based thermal proxies. We explore these twelve options in different combinations in the sections that follow. Inevitably, there are always further axes one might consider. (For example, simultaneous multithreading and heterogeneous cores are two other axes which impact thermal issues.) Nonetheless, we feel that this taxonomy helps guide us through many interesting and useful combinations of thermal design features for CMPs.

4.2.2 Stop-go vs. DVFS

One of the most basic forms of dynamic thermal management is known as global clock gating [9] or "stop-go". This involves freezing all dynamic operations and turning off clock signals to freeze progress until the thermal emergency is over. When dynamic operations are frozen, processor state including registers, branch predictor tables, and local caches are maintained, so much less dynamic power is wasted during the wait period. Thus stop-go is more like a suspend or sleep switch rather than an off-switch.

Our stop-go mechanism is a coarse-grain operation signaled by the processor and carried out by the operating system. Once a thermal sensor reaches the designated threshold, a thermal trap is signaled and processes are frozen for 30 milliseconds. After lowering the temperature a few degrees through stalling, the processor can resume. We choose this interval to be coarse-grain in part because it reflects the slow heating and cooling time constants of thermal variations (milliseconds [28]), and in part because it leads to a relatively simpler implementation.

The DVFS policy involves more of a continuous adaptive scheme. By enabling a continuous range of frequency and voltage combinations we can predictively use these to reduce power consumption. Thus our DVFS policy is not as simple as the stop-go mechanism, but we leverage past control-theoretic work to systematically obtain suitable parameters. We use a setpoint slightly below the thermal threshold and use a PI controller to adaptively control the frequency and voltage levels to aim towards this target threshold. Our DVFS mechanism has a higher design cost than the rudimentary stop-go mechanism due to the complexity of implementing a flexible phase-lock loop (PLL) and voltage scaling capabilities.

4.2.3 Distributed Policies vs. Global Control

While our first axis of classification in Table 4.1 focuses on the decision of stop-go vs DVFS policies, our second axis designates the scale on which these policies are applied.

One possibility ("global") is to implement a stop-go or DVFS policy regarding the entire chip as a single unit. This has been the method used primarily in the first generation of commercial multicore processors, due to its reduced design complexity. In the case of DVFS, this avoids communication difficulties that would arise with multiple clock domains. Furthermore, if all cores are likely to heat at the same rate, a policy which cools all cores in sync could be sufficient.

While global control policies work well for workloads that heat the chip uniformly, our real-system measurements from Chapter 1 that this uniformity is relatively unlikely. "Performance asymmetry"—the likelihood of workloads showing very different performance characteristics depending on the choice of applications—is a clear characteristic of emerging multicore applications [5]. If a global policy is used, a single hotspot on *one* of the cores could result to unnecessary stalling or slowdown on *all* cores. The more cores on the chip, the more potential performance is lost due to the single hotspot. Distributed policies, such as "Dist. DVFS" and "Dist. stop-go" as labeled in Table 4.1, instead allow each core to independently handle its own thermal management to a good extent. This chapter shows that for thermal purposes, choosing a distributed policy may be well worth the necessary added design complexity.

4.2.4 OS-based Migration Controllers

The final axis we consider regards the migration policy. Migration can help balance heat production across all cores. All the previously-mentioned policies—and combinations of them—can still have remaining thermal imbalances which can be further remedied through migration. Consider for example a common case: in a 2-core system managed by a DVFS policy, the integer register file could be the limiting hotspot on one core, while the floating-point register file might be the limiter on the other core. Judiciously migrating threads can allow the system to achieve better performance than DVFS-based methods alone.

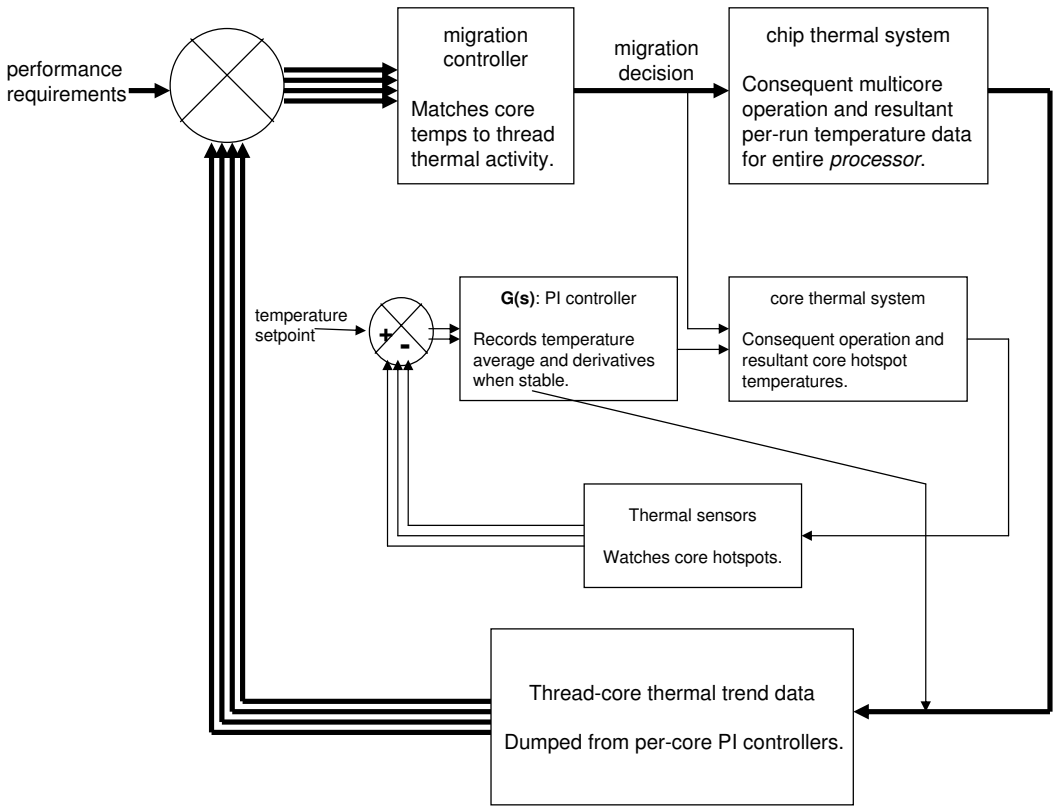


Figure 4.1: Feedback control system involving inner loop for local control and outer loop for coarse-grain migration decisions. The sets of four thick lines represent sets of data from each core.

We consider migration policies managed by the operating system. Timer interrupts from a typical OS happen on the order of a millisecond apart, and this is actually more than enough to get sufficient potential from migration. Particularly in our best cases of distributed DVFS combined with migrations, the hotspot "drift" is much slower than a typical temperature gradient in the more thermally-chaotic stop-go policy. Given this time scale, when migration is used on top of control-theoretic DVFS, we can model our overall system as a two-loop structure as shown in Figure 4.1. The inner loop is the DVFS policy, while the outer loop is the migration policy. The migration policy depends to a great extent on data gathered by the PI controller in the inner loop.

We study two migration mechanisms in this chapter. The first one is based on

performance counters used to determine the resource intensities of various threads. We draw some of these ideas from earlier works [19, 66], which describe concepts of mixing complementary resources based on profiled information. Our second mechanism is known as sensor-based migration. The purpose of this mechanism is to avoid reliance on performance counter proxy data which may be too abstract at times. Although both migration mechanisms, as well as all DTM policies in our study, rely on thermal sensors to make proper decisions at the correct times, the difference between the counter-based and sensor-based migration policies is that the latter uses sensors over time to track thermal properties of all processes. An elegant property of the sensor-based approach is that it depends directly on data gathered from the inner control loop as depicted in Figure 4.1.

4.3 Simulation Methodology and Setup

This section details our architectural, power estimation, and temperature modeling infrastructure. Figure 4.2 shows the overall flow. This section describes each of the illustrated levels of modeling, our modifications for thermal control, our test benchmarks, and metrics used.

4.3.1 Turandot and PowerTimer Processor Model

Using Turandot [60] we model a 4-core processor as detailed in Table 4.2. Most internal core parameters are similar to those used in our earlier work [19] and a study by Li et al. [51], although for example, we use a larger (4 MB) L2 cache. The overall configuration of each core of this processor may be thought of as a generation beyond the single-core SMT processor used in Chapter 3. While having many of the same functional units as the SMT processor, each core is modeled with a faster clock rate (3.6 GHz instead of 1.8 GHz) and a floorplan of smaller proportions (explained further

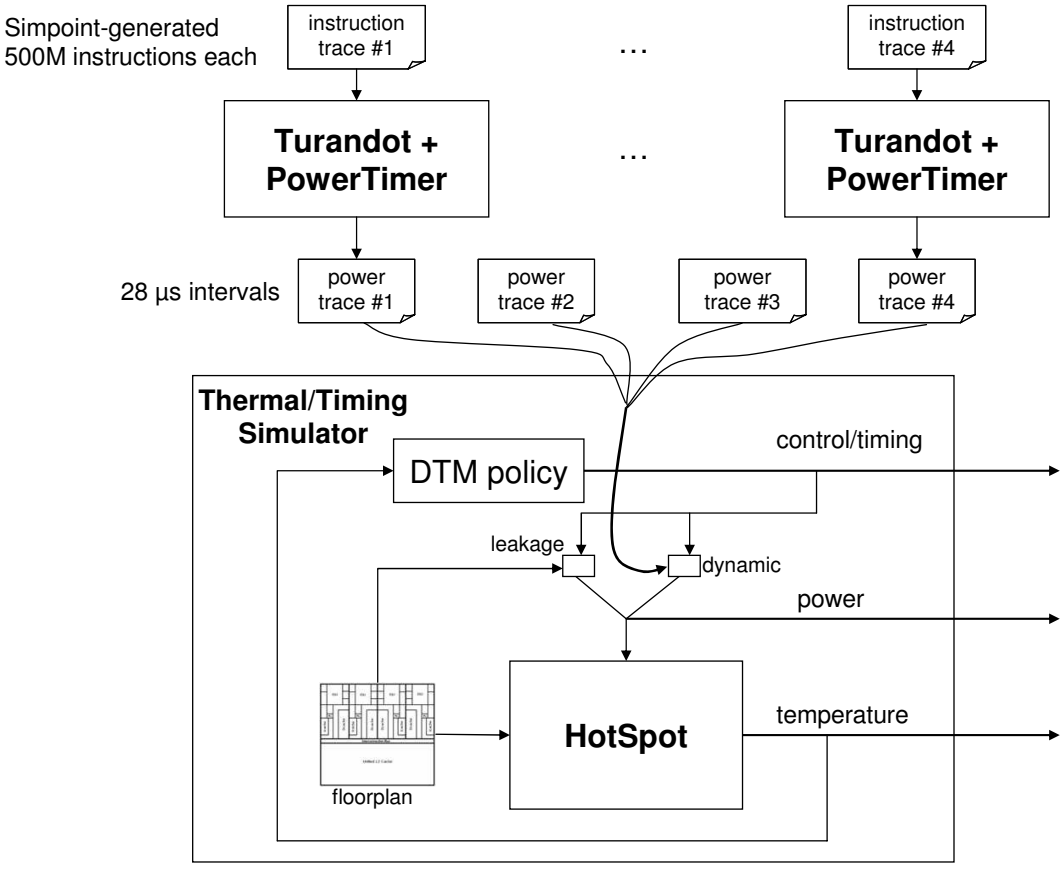


Figure 4.2: Power trace-based simulation method involving interdependence between Turandot, PowerTimer, leakage modeling, HotSpot, and thermal management policies.

in Section 4.3.2).

PowerTimer [8] is a parameterizable power estimation tool which operates in conjunction with Turandot. Its hierarchical power models are derived through empirical circuit-level simulations and calibrated according to technology parameters. Component power across simulation intervals is then calculated by scaling according to the counts of various architectural events. As shown in Figure 4.2, we use Turandot and PowerTimer to generate power *traces* to be used as inputs to our thermal simulator. Using SimPoint [70], we simulate representative traces of 500 million instructions from each program. These produce long (hundreds of milliseconds) output traces of power behavior containing data samples every 100,000 cycles (28 μs).

Global Design Parameters	
Process Technology	90 nm
Supply Voltage	1.0 V
Clock Rate	3.6 GHz
Organization	4-core + shared L2 cache
Core Configuration	
Reservation Stations	Mem/Int queue (2x20), FP queue (2x5)
Functional Units	2 FXU, 2 FPU, 2 LSU, 1 BXU
Physical Registers	120 GPR, 108 FPR, 90 SPR
Branch Predictor	16K-entry bimodal, 16K-entry gshare, 16K-entry selector
Memory Hierarchy	
L1 Dcache	32 KB, 2-way, 128 byte blocks, 1-cycle latency
L1 Icache	64 KB, 2-way, 128 byte blocks, 1-cycle latency
L2 cache	4 MB, 4-way LRU, 128 byte blocks, 9-cycle latency
Main Memory	100-cycle latency
DVFS Parameters	
Transition penalty	10 μs
Minimum freq scale	20% (720 MHz)
Minimum transition	2% of range
Migration Parameters	
Migration penalty	100 μs

Table 4.2: Design parameters for modeled CPU and its four cores.

Leakage power is becoming a significant component of total power, especially with more aggressively-scaled technologies. We cannot rely on PowerTimer, however, for leakage values since these numbers are dependent on temperature, whose calculation comes later in our toolflow. Therefore, we describe our leakage power modeling approach below, as part of the thermal/timing approach.

4.3.2 HotSpot Thermal Model

As a component of our thermal/timing simulator, we use HotSpot version 2.0 [35, 74] which uses parameters that have been calibrated against a real chip as well as a

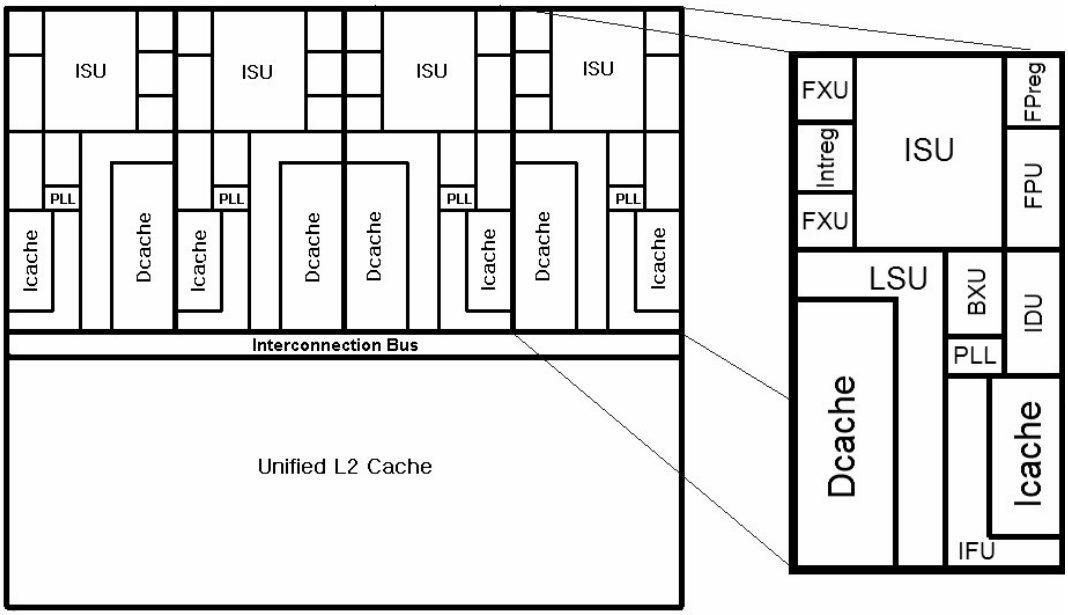


Figure 4.3: High-level floorplan input to HotSpot 2.0 showing our 4-core CMP and a close-up view of the core components.

power-trace-driven simulation capability. HotSpot calculates temperatures by modeling physical traits in a thermal system using a method analogous to calculating voltages in a circuit made up of resistors and capacitors. Required inputs to our thermal simulator include a floorplan designating the locations and adjacencies of various processor components. The model also includes the heatsink, fan (convection), and thermal interface material.

We use a floorplan similar to that used in a study by Li et al. [51], except we have extended our layout for 4 cores and reduced the core size accordingly. Each of these cores has its various components necessary for an out-of-order pipeline and the four cores are connected through a shared L2 cache and interconnect. Our overall floorplan, alongside a detailed floorplan of the core, is shown in Figure 4.3. Compared to the core floorplan used in Chapter 3, this core floorplan has been shrunk in the horizontal direction while still maintaining the same basic organization of functional units.

HotSpot supports calculating transient temperatures as well as estimating steady-state temperatures. A number of past works [18, 27, 31, 74] have focused on steady-state. One advantage of steady-state temperatures is the ability to estimate long term temperatures from only a short simulation interval. Our experiments in adaptive control, however, require our simulator to know how temperatures change across time. Hence we focus on transient temperatures.

4.3.3 Thermal/Timing Simulator for DTM

As depicted in Figure 4.2, our thermal/timing simulator tests our thermal management policies in order to collect timing, power, and temperature data for any of our multiprogrammed workloads. The thermal/timing simulator uses power traces as inputs to its thermal control simulations. The simulator uses these recorded power values, controls the rate of progression through the trace, and scales power values in response to thermal control decisions. With DVFS, for example, it adjusts the time and energy calculations for that core (or for the whole chip) to account for the new voltage/frequency setting. Because DVFS dynamically changes the length of a cycle, and because in some of our methods each core may be operating with a different cycle time, the thermal/timing simulator framework tracks progress in terms of "absolute time" rather than cycles.

When a power trace for a particular benchmark is completed before the end of the simulation, that trace is restarted at the beginning and this process is continued until total of 0.5 seconds of silicon time has elapsed. For calculating leakage dynamically, we use the temperatures reported by HotSpot as input to a leakage model based on an empirical equation from [34].

In order to model shared structures such as the L2 cache with this trace-based method, the single-threaded simulations with Turandot actually are capacity-limited to use only one quarter of the L2 cache, while retaining pessimistic power costs of

the full-size cache. This likely overestimates cache power, but the cache is never a hotspot. Another pessimistic approximation result of our methods is that for a DVFS mechanism. A memory-bound application can exploit CPU-memory slack [87], which our traces will not show. Thus, it is likely to gain more energy savings from DVFS compared to performance loss than a CPU bound application.

Overall, the benefits of this coarse-grain power trace method are (i) that it simulates dynamic thermal variations and (ii) that it allows us to apply our control methods on the long time scales appropriate for observing temperature changes. After initial publication of our work, a variation of our coarse-grained approach was used by Chaparro et al. [14]. They also showed that at the cost of having to generate several times as many traces, it is possible to better model the performance effects of frequency scaling. It may also be possible to address other issues, such as cache capacity modeling, with more complex tracing techniques.

4.3.4 Workloads

Our simulated workloads are formed by selecting among 22 benchmarks including eleven SPECint benchmarks and eleven SPECfp benchmarks to mix into designated four-process workloads as shown in Table 4.3. As was the case in Chapter 3, each benchmark's type is especially relevant to our study on overheating specific resources. Integer benchmarks are most likely to have their prime hotspot in the integer register file unit and its associated logic while floating point benchmarks are likely to stress the floating point register unit. Thus, we list the corresponding benchmark suite categories for all elements of each workload.

The type of benchmark alone—whether integer or floating point—does not completely categorize its resource intensities. Integer benchmarks in the SPEC suite have varying degrees of IPC. Furthermore, all floating point benchmarks make use of integer registers to some extent. Some floating point benchmarks even have higher

Workload name	Benchmarks	Properties (integer / floating point)
workload1	gcc, gzip, mcf, vpr	Int, Int, Int, Int
workload2	crafty, eon, parser, perlbmk	Int, Int, Int, Int
workload3	bzip2, gzip, twolf, swim	Int, Int, Int, FP
workload4	crafty, perlbmk, vpr, mgrid	Int, Int, Int, FP
workload5	gcc, parser, applu, mesa	Int, Int, FP, FP
workload6	bzip2, eon, art, facerec	Int, Int, FP, FP
workload7	gzip, twolf, ammp, lucas	Int, Int, FP, FP
workload8	parser, vpr, fma3d, sixtrack	Int, Int, FP, FP
workload9	gcc, applu, mgrid, swim	Int, FP, FP, FP
workload10	mcf, ammp, art, mesa	Int, FP, FP, FP
workload11	ammp, facerec, fma3d, swim	FP, FP, FP, FP
workload12	art, lucas, mgrid, sixtrack	FP, FP, FP, FP

Table 4.3: Four-process workloads of interest and respective mix types of SPEC benchmarks.

integer register intensities than various integer benchmarks, as seen in Chapter 3. Nonetheless, the general categorization is a helpful guide in understanding how such workloads might behave subject to varying forms of thermal control.

4.3.5 Metrics

Our goal in these thermal control applications is to maximize performance subject to a fixed temperature constraint, in our case not allowing any part of the chip to go above 84.2° C. Although this threshold was arbitrarily chosen since it matched one of our well-tuned controllers, it is also possible to use any other threshold and adapt the control policies accordingly. Under this constraint, one of the most natural performance metrics is the raw instruction throughput for each workload (Billions of instructions per second, or BIPS).

While BIPS is a good basic performance metric, it can, however, be difficult to interpret in multiprogrammed workloads. This is because fairly running a low-IPC application can lead to worse-appearing performance than unfairly skewing execution toward the high-IPC applications in the workload. For this reason, we also provide

results on each run's achieved percentage of "duty cycle". Duty cycle describes the ratio of time that useful work is being done, relative to the total time including work period and the rest (stop) period [66]. For example, if a processor is in a globally-stalled mode four times as often as it runs in active mode, it has a duty cycle of 20%.

While duty cycle is very straightforward for stop-go situations, we also adapt it to DVFS as well. Although the processor may attempt to do useful work at each cycle, the varying frequencies in DVFS approaches change the effective duty cycle. For this reason, we use an adjusted duty cycle metric as follows. When summing up the total duty cycle, we scale the contributions accordingly by the dynamic frequency. For example, if all cores run at 30% of maximum speed for an entire execution this amounts to a duty cycle of 30%. If all cores run half the time at 30% speed and the other half of the time at 40%, this results in a duty cycle of 35%. This adjusted duty cycle is a good indicator of the ratio of the total work done relative to the total possible work that could be done if all cores were run at their maximum clock frequency not subject to any thermal constraint. Under this reasoning, overhead delays (such as that for adjusting the PLL under a DVFS policy) or for the context switch penalty under a migration policy are not counted as useful total work and thus also lower the adjusted duty cycle.

4.4 Applying Formal Control to Thermal DVFS

The DVFS portion of our thermal-management mechanisms use a control theoretic approach to determine appropriate voltage and frequency settings. Here we present some background information required to understand this approach, before introducing the other policies and our comparative results.

4.4.1 Background: Closed-loop DVFS Control

When designing our DVFS controller we apply closed-loop control theory. Formal feedback control has recently found numerous applications in architecture and systems [73, 77, 86, 87, 88]. Our work is novel, however, in composing together these formal control methods along with other control techniques in a multi-loop system.

Closed-loop control is a robust means to control complex systems so that the controlled value rapidly converges to the desired target output value. In our case, the measured variables are the thermal sensor values at various hotspots, and the output actuator is the mechanism to scale the voltage and frequency. This control loop is represented in the inner loop of Figure 4.1. Some of the arrows in a typical closed-loop diagram have been drawn as several arrows in Figure 4.1. This reflects that multiple temperatures are fed into a single PI controller. Since an individual controller governs an entire core or processor, it typically selects the hottest of the input temperatures.

The standard PI controller equation, written in its Laplace form, is as follows.

$$G(s) = K_p + \frac{K_i}{s}$$

As shown, the two components are the proportional and integral terms. The proportional component defined by K_p reflects the basic gain of the controller responding to error, while the integral term containing K_i is there to compensate for any offsets and reduce the settling time. (A proportional-integral-derivative (PID) controller is another option, but we found that the derivative term has little benefit for this type of thermal control.)

MATLAB tests similar to those applied by Skadron et al. [73] allow us to determine settling time and stability for typical thermal fluctuations. We use constants of $K_p = 0.0107$ and $K_i = 248.5$ in all of our tests. Owing to the robustness of PI

systems and the inherent stability of the thermal system under study, these constants can actually deviate significantly while still achieving the intended goals. In fact, our proportional constant is set two orders of magnitude smaller than the configuration used by Skadron et al. [73], in order to maintain control with smoother transitions. Another issue is that of the system's sensor delay. Fortunately, the primary delay (thermal sensors) is quite small [16] compared to the time scales on which temperature varies.

A benefit of formal feedback control is the ability to prove stability. On a root locus plot, the stability criterion is that all the poles (the frequencies at which the characteristic function blows up to infinity) must lie to the left of the y-axis in the Laplace space. We verified this for our controller using MATLAB.

4.4.2 Thermal Control Mechanism for DVFS

The mathematical description given above proposes a controller that can be applied continuously, not limited by physical constraints. In our actual experiment this controller must be given appropriate limits and be discretized in time. In order to convert the Laplace transform into its corresponding discrete-time z-transform, we use the MATLAB function `c2d`, specifying a time interval of 28 μs to match the frequency of our thermal measurements. We then arrange the transform to directly specify our discrete online equation:

$$u[n] = u[n-1] - 0.0107e[n] + 0.003796e[n-1]$$

The error function $e[n]$ is simply the difference between the measured temperature and the target temperature. The target temperature is just below the thermal threshold. Through this, the system maximizes performance subject to the allowable temperature constraints. Convenient aspects of this controller are that despite involving an

integral term, it is relatively simple to implement in hardware as it depends only on the previous controller output, previous error value, and current error value.

On a real system, PI controllers are subject to certain limits which stray from a purely linear design. One of the most basic limits is that of clipping on the output. The output, which is the specified frequency scaling factor, cannot extend to infinity since the cores cannot run beyond their maximum frequency as limited by the speed of transistor gates. When a core or processor is not in thermal danger but rather acting in a cool period, the controller will output its maximum value which reports a frequency scaling factor of 1.0. On the lower end, we restrict the minimum frequency scaling factor to 0.2. In the discrete model described in the above equation, this can be implemented fairly simply in hardware.

Another non-ideal characteristic of real PI controllers is that of integral windup, which describes when the integrator component continually integrates only because the input error remains present for an extended period due to physical limits. For instance, when a core is above its target temperature, the controller will try to cool it by lowering the frequency. However, chip components cannot cool any faster than the physical limitations allow. If integral windup occurs, when the condition is finally satisfied it can take a long time for the controller to "wind down". Fortunately, our discrete implementation with clipping quickly takes care of this. The simple discrete implementation in the above equation combined with clipping prevents a hidden integral component from building up.

Finally, in real systems, voltage changes are not instantaneous. A penalty of $10\mu s$ is assumed for each frequency and voltage change.

4.5 Exploring Stop-go and DVFS in both Global and Distributed Policies

This section covers a portion of the policy combinations in our spectrum. In particular, we examine issues of using stop-go and DVFS, and we also consider whether to apply each mechanism in a local or distributed fashion. Section 4.6 then explores migration policies largely with the intention of comparing to the original policy combinations in this section.

4.5.1 Stop-Go Policy Implementations

Compared to the formal control approaches used for DVFS management as described in the preceding section, our stop-go mechanism is quite simple. Each core is run at full blast as long as it does not exceed a particular thermal trippoint. Thermal sensors at the two register file units on each core sense the hotspot temperatures. When one is found to be just below the thermal threshold of 84.2° C, a thermal interrupt is issued. The core which caused this interrupt is then stalled for 30 ms. At this point, the hotspot will have cooled below the threshold and the core can continue running.

4.5.2 Distributed versus Global Policy Implementations

In the distributed policies, stop-go and DVFS techniques are applied to individual cores. Each DVFS controller takes in at least two inputs since it watches two hotspots, but the mathematical implementation goes by whichever sensor reports to be hotter. In the distributed cases, each core operates independently, without any coordination with other cores. For the cases of global stop-go and global DVFS, a single decision is made for all the cores on the chip. Thus, there is effectively only a single PI controller which calculates based on the hottest of all sensors across all cores.

	BIPS	duty cycle	relative throughput
Stop-go	2.79	19.77%	0.62
Dist. stop-go	4.53	32.57%	1.00
Global DVFS	9.36	66.49%	2.07
Dist. DVFS	11.36	81.02%	2.51

Table 4.4: Average instruction throughput, effective duty cycle, and performance relative to dist. stop-go across all workloads for various policies.

4.5.3 Experimental Results

Results for all twelve workloads are shown in Figure 4.4. We report results relative to a baseline policy of distributed stop-go. Global stop-go has the worst performance of the twelve. Its duty cycle is less than 20% and for this reason, we focus our attention on the other eleven possibilities for the remainder of the chapter. Although our baseline characteristics are heavily dependent on our processor configuration and cooling system model, we have examined other scenarios. For example, we found that raising the temperature threshold to 100° C increased the duty cycles of these results and others presented in our chapter by 10 to 15%. Nonetheless, the relative performance tradeoffs remain as presented.

We present instruction throughput for the three non-migratory configurations—global stop-go, synchronous DVFS, and distributed DVFS—all normalized to the distributed stop-go results. As seen in the graph, the distributed DVFS policy does the best overall, on average more than double the instruction throughput of the distributed stop-go policy and four times that of global stop-go. The largest gain is due to voltage scaling. In addition, however, the distributed DVFS policy still gives significantly better throughput than the global. Tabulated results showing the average throughput and duty cycle for these policies are given in Table 4.4.

The overall improvement by distributed DVFS over a distributed stop-go policy is a performance improvement of 2.5X. The duty cycle numbers also reflect these

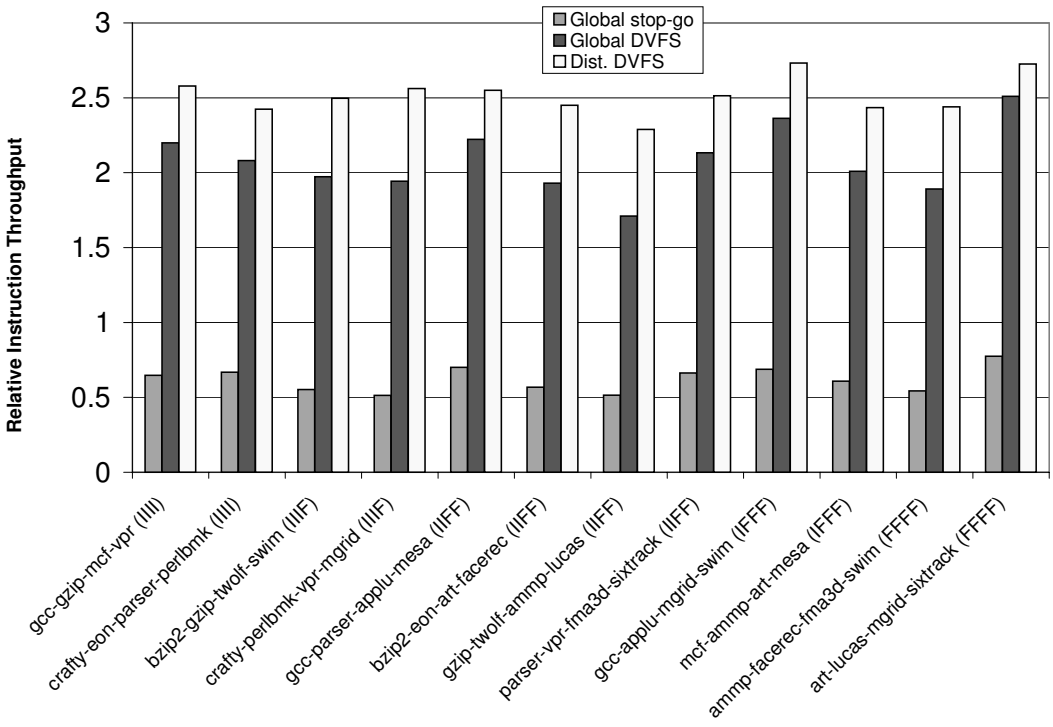


Figure 4.4: Normalized instruction throughput of all workloads relative to baseline distributed stop-go policy.

improvements. In particular the active time reported for distributed stop-go is about 30% while distributed DVFS achieves more than 80%. We also have performed other experiments confirming the validity of our duty cycle metric. We ran simulations with unrestricted maximum temperatures, and found that the proportion of the achieved BIPS relative to the non-controlled case was accurately predicted by the measured duty cycle.

The benefits seen here are consistent with past work which has shown much benefit from DVFS policies as opposed to primitive throttling policies [9, 51]. Likewise, allowing multiple voltage levels is greatly beneficial for power [40] and hence thermal control. The drawback of these design choices is of course mainly in design complexity. There are certainly challenges in implementing proper coordination and communication among several cores running at different frequencies. Furthermore, to enable multiple voltage levels, each core would need separate voltage regulators

which implies additional costs in terms of area and complexity. As our results show, however, for high-performance processors under thermal stress, the design costs of the distributed DVFS are reasonably rewarded by the clear benefits of improved performance.

4.6 Migration Policies for Thermal Control

The third axis in our spectrum of thermal control options is determined by whether or not to use migration and the choice of migration mechanism used. We explore migration mechanisms implemented via OS control for two main reasons. First, benefits from migration policies happen on a relatively long time scale if migration is implemented in combination with policies that work on a finer-grain time scale such as stop-go and DVFS. Second, process control and context switches are traditionally something for which the operating system has final jurisdiction. Thus our migration mechanisms are called upon no more than once every 10 milliseconds, which is the typical timer interrupt setting for a Linux kernel. Both of our migration mechanisms, counter-based and sensor-based, are affected by the DVFS or other policies previously explored and thus a feedback relation exists when both DVFS and a migration policy are implemented. In this relation, the operating system needs to keep track of timing data for all processes. Although not explored in our experiments which are restricted to four-program workloads, in any system there can easily be a greater number of processes than cores.

When the OS decides to migrate threads for the purpose of thermal control, the relevant tracking information is flushed and stored and in our simulations, each core involved takes a penalty of 100 μs. Once this is completed, the overriding thermal policy is the primary mechanism of thermal protection, until 10 milliseconds have passed and the involved threads become eligible for migration again. The actual

decision algorithms for which threads to migrate are described in more detail below.

4.6.1 Counter-based Migration: Method

We examine first a performance counter-based migration policy, which is the simpler of the two discussed. The counters are used here to estimate the thermal intensity of a particular resource. For example, for the integer register file, performance counters are used to keep track of the number of accesses for individual threads. The performance counter information used here includes cycle counts, the number of integer register file accesses, the number of floating point register accesses, and instructions executed. Much of the reasoning for our mechanism is borrowed from the work by Powell et al. [66], which proposes mixing resource heat intensities with SMT. With no frequency scaling, we are interested in the ratio of register file accesses per cycle. When frequency scaling is used, it becomes necessary to know the number of accesses per adjusted cycle instead.

Our overall algorithm is shown in pseudocode in Figure 4.5. Every thread's various performance counters are recorded throughout execution. This way the operating system is aware of the various resource intensities of all running programs. Migration decisions are actuated when the local thermal control of at least two individual cores signals that their critical hotspots have changed. If this happens more often than 10 ms, extra requests are simply ignored since there is little reason to enact a thermal migration on such a short time scale. When it is time to test for eligible migrations, the cores with the most critical hotspot imbalance are considered first. Hotspot imbalance is defined as the difference in temperature between the core's critical hotspot and the temperature of the core's second hottest distinct hotspot. In order of most need for migration, the decision algorithm searches for a suitable candidate. Each decision is done by seeing which thread would be most able to reduce heating of the critical hotspot on each core. In some cases, the best candidate for a thread to migrate will

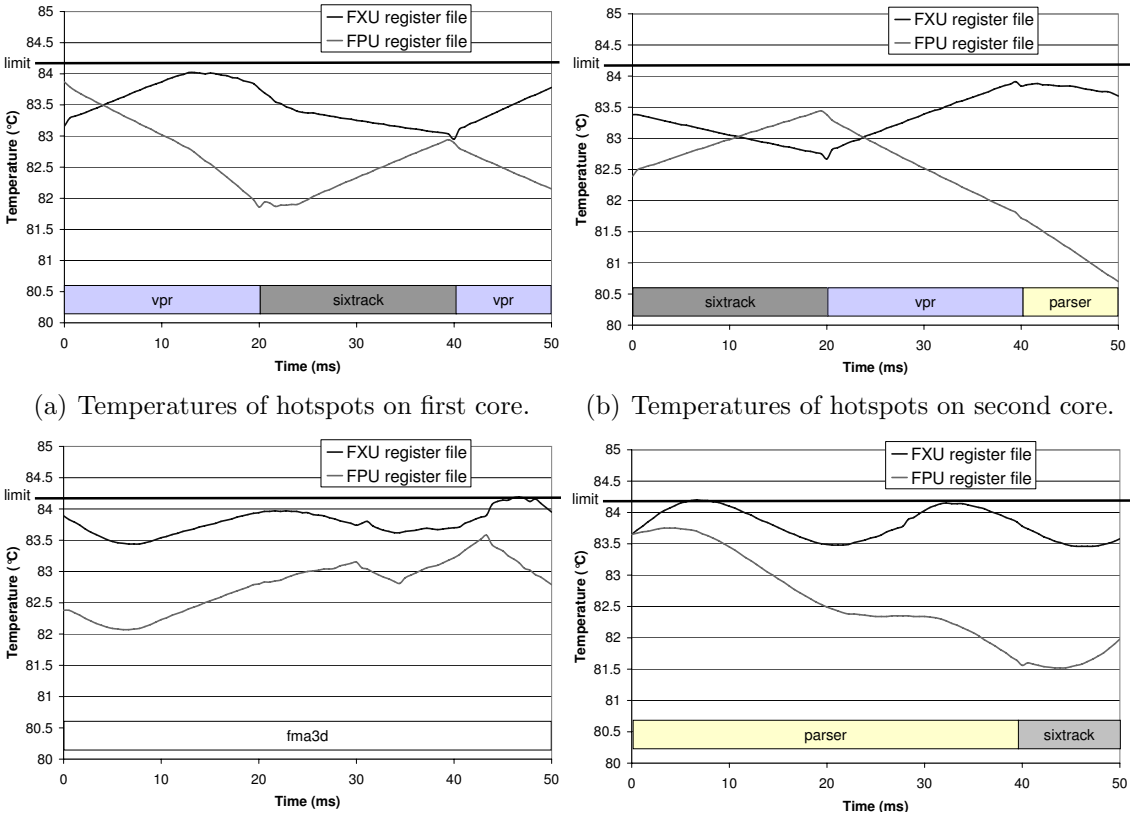
```
(1) remaining_processes = processes[];
(2) sort(cores[], most_hotspot_imbalance)
    where hotspot_imbalance =
        critical_hotspot.temperature -
                secondary_hotspot.temperature for core[i];
(3) foreach (cores[1..n]) { // find best matchings
    matching_process = least_intense(remaining_processes,
                        cores[i].critical_hotspot);
    cores[i].assigned_process = matching_process;
    remaining_processes -= matching_process;
  }
(4) foreach (cores[1..n]) { // migrate if beneficial
    if (cores[i].current_process != cores[i].assigned_process)
        migrate(cores[i].assigned_process, cores[i]);
  }
```

Figure 4.5: Algorithm pseudocode for thread-to-core matching for migration decisions, both for counter-based and sensor-based migration.

be itself, in which case a migration is not done. A set of migrations can be as simple as a single swap, or as complex as a four-way rotation.

4.6.2 Counter-Based Migration: Results

Figure 4.6 (a) shows the effects and intentions of several migrations on a single core using `parser-vpr-fma3d-sixtrack` as an example. Initially, `vpr` is running on this core, but after 20 ms it is switched out so that a floating-point intensive benchmark, `sixtrack` can migrate in and bring the two hotspots closer to balance. The next migration occurs at time $t = 40$ ms, in which `vpr` returns to this core. This migration does not appear necessary for the core by itself, but its cause can be better seen in the context of the other three cores. Figures 4.6 (b) through (d) shows the hotspot temperatures of the other four cores. As seen in Figure 4.6 (d), the fourth core began to use `sixtrack` at $t = 40$ ms, which is why this benchmark was migrated away from the first core. Figure 4.6 (b) also shows the second core balancing the two hotspot temperatures, and even managing to bring the FPU register file to the hotter



(a) Temperatures of hotspots on first core.

(b) Temperatures of hotspots on second core.

(c) Temperatures of hotspots on third core.

(d) Temperatures of hotspots on fourth core.

Figure 4.6: Temperatures of key hotspots across three migration periods for `parser-vpr-fma3d-sixtrack`. At the base of each graph are blocks denoting which benchmarks are present (migrated in) across sections of the time axis.

of the two hotspots at one point. On the third core, `fma3d` is never migrated away throughout the entire 50 ms interval. The reason for this is that `fma3d` is reasonably balanced with close levels of activity on both the integer and floating point register files.

Our overall results from this counter-based migration policy are shown in Table 4.5. When used in conjunction with a distributed local stop-go policy, counter-based migration provides a 2X performance improvement and sees about the same increase in duty cycle. This means that in cases where DVFS is not available, stop-go can be used for basic heat reduction, and migration provides better performance through heat balancing. In Table 4.5 we have also included the frequency of migrations for

	BIPS	duty cycle	relative throughput	migrations/core /second	speedup over non-migration
Stop-go, counter-based migration	5.34	37.93%	1.18	57.4	1.91
Dist. stop-go,counter-based migration	9.15	65.12%	2.02	51.9	2.02
Global DVFS, counter-based migration	9.88	70.05%	2.18	52.7	1.06
Dist. DVFS, counter-based migration	11.62	82.42%	2.57	46.5	1.02

Table 4.5: Average instruction throughput and duty cycle for performance counter-based migration policies.

each policy combination. However, because migrations happen infrequently enough for their direct performance cost to be small, there is not necessarily any direct correlation between the overall performance of different policies and their frequencies of migration. In the next section, we discuss our alternative policy that does not depend on performance counters.

4.6.3 Sensor-based Migration: Method

The counter-based approach is appealing because it relies on easily-accessed hardware counters of microarchitectural activities that have intuitive meaning to hardware and software designers. Furthermore, their values can be directly attributed to threads and code. They are not, however, a direct representation of thermal behaviors. Instead, they are at best a proxy.

Here we explore instead a second migration method based directly on reading on-chip thermal sensors. With sensor-based policies, the mechanism is complicated significantly by external factors. Although vertical heat conduction typically matters more than lateral heat conduction [18, 27, 74], the lateral effects are not small enough to ignore. Our sensor-based mechanism seeks to know the slopes of temperature transitions, and these may depend on hotspots from a neighboring unit or core. For example, a certain thread will appear to have different temperature gradients when running on different cores due to different external factors, such as being located closer to the edge of the chip. Furthermore, if a DVFS or stop-go policy is applied, the trend

sensing calculations must appropriately time-scale the measured temperature changes to account for this. Depending on which core and at what time measurements are taken, this could give different results. Because of these issues, our design requires recording the scaling factors (as seen by the PI controller) and using the average to scale the measured thermal trends appropriately.

Our algorithm for the sensor-based migration is more complex than the counter-based policy. Although our decision algorithm is almost the same as that presented in Figure 4.5, determining individual threads' hotspot intensities through thermal sensors is more complex than direct counter information. The apparent intensity on various cores for a single thread will appear different as each core has different thermal situations. For instance, a core next to the cache may have less thermal intensity due to the cache's relatively cool temperature. We therefore need to profile threads in a systematic manner so that relative temperature gradients can be used to estimate the thermal intensity of all possible thread-core combinations. The flow diagram in Figure 4.7 describes our steps to accomplish this. There is a grid maintained by the operating system so that the migration decision can be made to estimate a thread's hotspot behavior on a particular core. To estimate thread intensity, each core needs to be run and dynamically tested with at least two threads, and each thread needs to have recorded sensor data from running on at least one core.

A benefit from this approach is that much of the feedback information can be recorded in the PI hardware which does arithmetic operations with the temperatures on a time scale appropriate for recording the trends. As with counter-based migration, for the distributed DVFS case, each recorded temperature trend must be scaled down by a cubic relation according to the recorded frequency scaling factor.

Figure 4.8 demonstrates the flow of how the OS-managed data table can be filled and retained. The example given depicts the workload `gcc-gzip-mcf-vpr` going from a cold start all the way to the continuous thermally adaptive migration mode. After

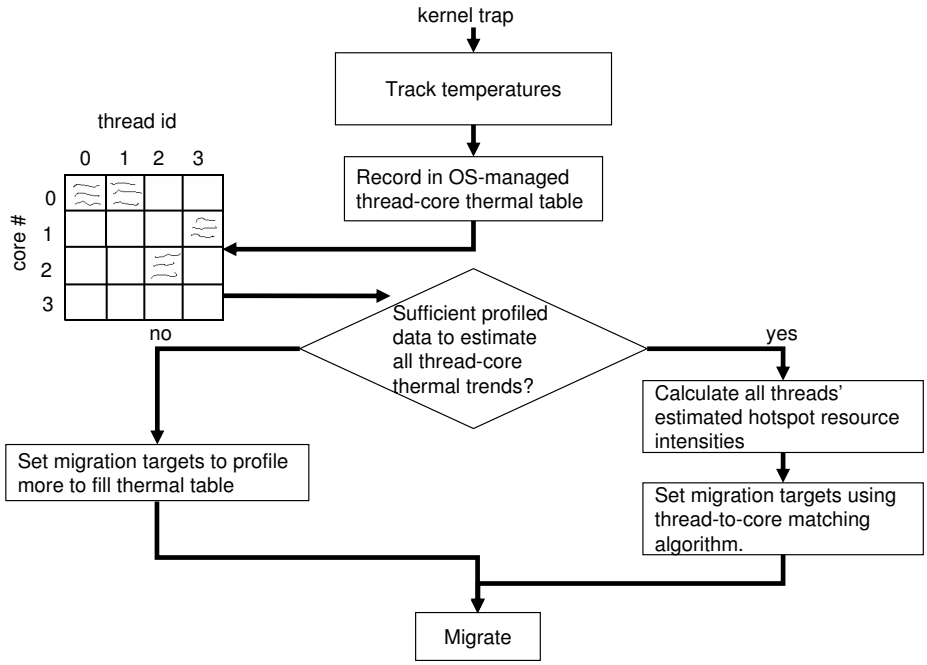


Figure 4.7: Flow chart demonstrating the steps taken upon an OS interrupt to decide on sensor-based migrations.

	BIPS	duty cycle	relative throughput	migrations/core /second	speedup over non-migration	speedup over counter-based migration
Stop-go, sensor-based migration	5.43	38.64%	1.20	59.1	1.95	1.02
Dist. stop-go, sensor-based migration	9.27	66.61%	2.05	44.3	2.05	1.01
Global DVFS, sensor-based migration	9.63	68.37%	2.13	41.3	1.03	0.97
Dist. DVFS, sensor-based migration	11.70	82.64%	2.59	36.6	1.03	1.01

Table 4.6: Average instruction throughput and duty cycle for sensor-based migration policies.

each migration interval, the thermal data table is updated accordingly. We have used different shades of gray to highlight which data entries are newly written at the end of each migration interval. After two intervals, the profiled thermally data is sufficient to always migrate using the algorithm given in Figure 4.5. Furthermore, as the thermally adaptive migration continues, the thermal data table continues to receive updates.

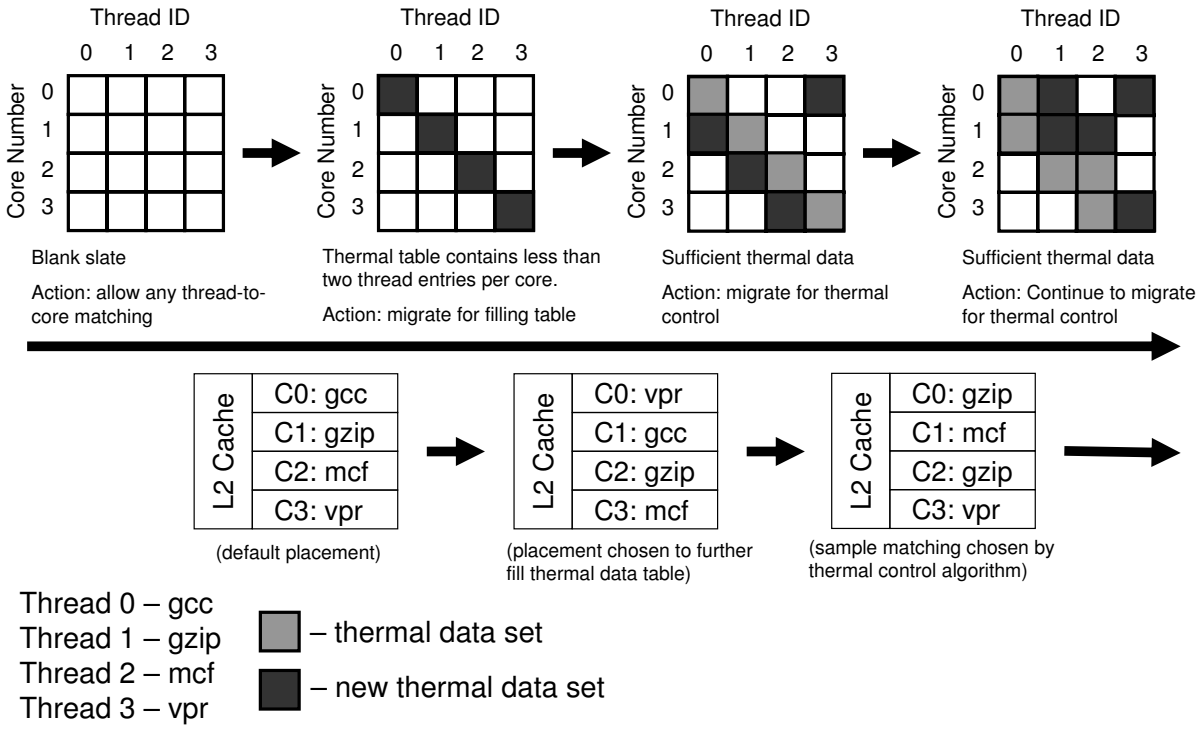


Figure 4.8: `gcc-gzip-mcf-vpr` as an example of the startup process for profiling then using thermal data for sensor-based migration.

4.6.4 Sensor-based Migration: Results

Tabulated results comparing the sensor-based migration policy with the non-migration and counter-based migration policies are compared in Table 4.6. For more detail on the most advanced policy combinations, we have plotted the individual workload performances across migration policies in Figure 4.9 comparing the performance effects of counter-based and sensor-based migration. Like counter-based migration, sensor-based migration provides noticeable speedup on most workloads. In some cases, such as the two all-integer workloads (`gcc-gzip-mcf-vpr` and `crafty-eon-parser-perlbmk`), both policies fail to create significant performance improvement. For these two workloads in particular, counter-based migration even slightly degrades performance. On average, however, both migration schemes slightly improve the overall performance when used in conjunction with distributed DVFS, and the hybrid policy of sensor-based migration with distributed DVFS does slightly better overall.

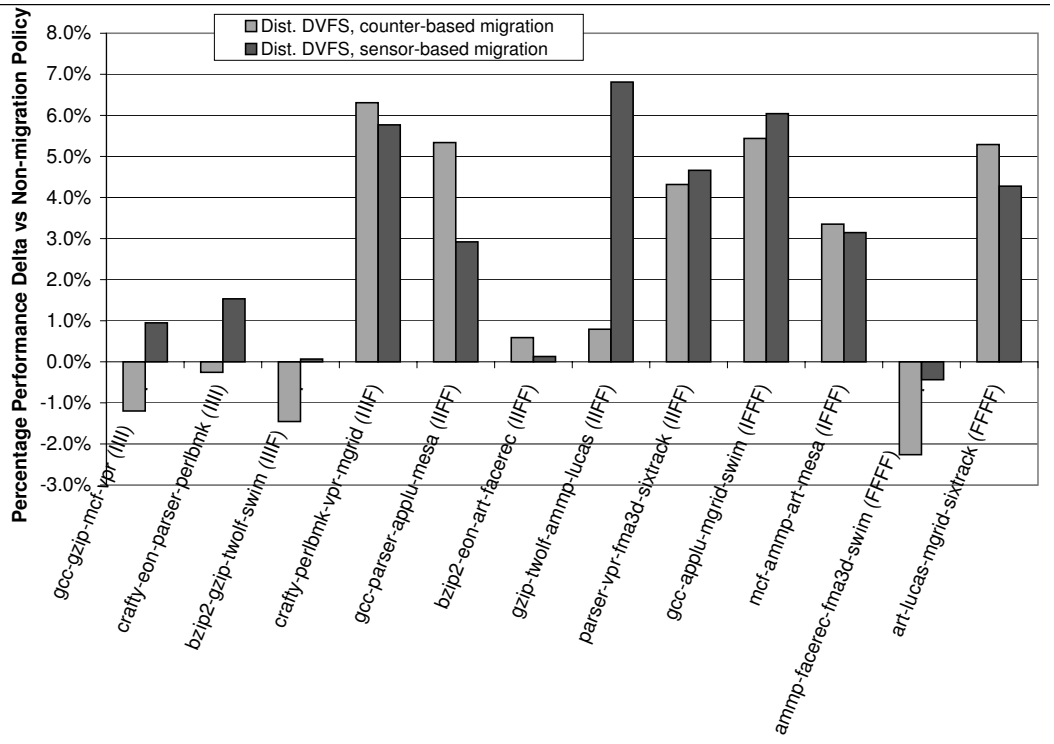


Figure 4.9: Individual gains/losses of various workloads due to either migration policy in conjunction with distributed DVFS (best-performing practical policy of the original four).

Our overall results show that both migration policies are beneficial and feasible. Sensor-based migration mechanisms perform with an overall 2.1X speedup over the baseline stop-go policy and an overall speedup of 2.6X with proportionally the same increase in duty cycle when combined with distributed DVFS. On the other hand, Figure 4.9 shows that neither migration policy succeeds in benefiting every possible workload. This shortcoming can be explained by the fact that they are both doing prediction algorithms to best estimate migration decisions. Errors due to the algorithm assumptions can lead to decisions that not always optimal, but on average do provide a significant performance benefit.

The options presented in this section and the previous section present designers with several viable choices for a total thermal management policy. In simpler designs such as global stop-go, migration makes up for much of the benefit that would be

	No migration		Counter-based migration		Sensor-based migration	
	Stop-go	**DVFS**	**Stop-go**	**DVFS**	**Stop-go**	**DVFS**
Global	0.62X	2.1X	1.2X	2.2X	1.2X	2.1X
Distributed	baseline	2.5X	2X	2.6X	2.1X	2.6X

Table 4.7: Summary of all policy combinations and their respective multiplicative increases in instruction throughput relative to distributed stop-go.

found in a system invoking DVFS. Furthermore, the migration controls are mostly OS-controlled and hence can be reconsidered and reprogrammed after chip production.

4.7 Results Overview

Instead of viewing some conventional mechanisms as competing alternatives, this chapter has explored a combination of orthogonal methods to determine where the most gains are seen and which policies work best together. To summarize this here, we recall our table from Section 4.2, and present a similar organization except filled with the overall relative instruction throughput for all policy combinations in Table 4.7.

Our basic results fortify distributed DVFS as a strong foundation for thermal control, reflecting on average more than 2.5X increase in throughput over our base policy of distributed stop-go.

Both counter-based and thermal trend-based migration policies are able to significantly increase performance through hotspot balancing, respectively reporting 2X and 2.1X improvements of the baseline stop-go policy. When implementing migration on top of other policies we see diminishing returns but a net benefit on the most aggressive distributed DVFS policy with a 2.6X speedup over baseline.

Duty-cycle measurements offer a good view of how close we have come to full-speed execution. For example, while simple stop-go techniques result in duty cycles

below 20%, the best multi-loop combination of migration and DVFS improves the duty cycle to an average over 82%. Given the threat of thermal emergencies, 100% duty cycle is not possible for these workloads under our experimental conditions, but values in excess of 80% are quite close.

4.8 Related Work

Since reducing power density has the effect of reducing temperature, temperature-aware approaches benefit much from the same techniques as in power-aware design. One key difference is that temperature-aware approaches seek not necessarily to reduce the average temperature but also focus on the thermal constraints of individual hot spots, as our work does. Other differences arise in the metrics that are most relevant. Our work uses many of these techniques demonstrated in prior studies on processor power, but applies these mechanisms directly to the problem of thermal control.

With temperature control as a key limitation to processor performance, many recent works in computer architecture focus on issues of thermal control [18, 19, 32, 35, 51, 52, 66, 74]. In particular, some of the more closely related works explore temperature-aware design issues in multithreaded architectures similar to ours. For example, our prior work [18] explored temperature issues in simultaneous multithreaded (SMT) and multicore designs and found common characteristics of thermal stress. We did not, however, delve into thermal control techniques to alleviate these problems. Ghiasi and Grunwald examine thermal properties of dual core designs in particular [27]. However, their work also focuses on steady-state temperatures for fixed configurations rather than dynamic control. Li et al. examine general issues of performance, power, and thermal characteristics of both multicore and SMT processors [51]. Although they do explore a number of thermal management policies,

they view these as competing alternatives rather than taking a broad approach like ours that includes combinations of techniques. Li and Martinez attempt to explore methods to model performance and power efficiency for multicore processors subject to thermal constraints [48], but they do not delve into the various options of thermal management policies. Chaparro et al. have examined issues on designing clustered processors and the potential for temperature improvement through clustering methods [13, 15]. While covering many microarchitectural details for such multicore designs, this work does not focus on the thermal control policy and uses a routine core-hopping mechanism.

Related more to our study are some other works which focus more on the design of control policies for thermal management. For example, Shang et al. have proposed adaptive mechanisms for thermal management by focusing primarily on interconnect power control, but they use techniques of a primarily power-aware nature rather than focusing on mitigating localized hotspots [69]. Powell et al. [66] describe techniques for thread assignment and migration and the intuitive nature of migrating computation appropriately to balance temperatures. Their work uses performance counter-based information in a similar manner to our counter-based migration policy. Although they compare their technique directly to stop-go and DVFS policies, they do not consider the possibility of combining such techniques as we have done.

Since the initial publication of this work, there have been further studies extending upon our covered design space. For example, Chaparro et al. have recently performed a similar exploration of migratory and distributive thermal control with as many as 16 cores [14]. Their results show that much of the intuitive results in this chapter are extensible to future technologies and greater numbers of cores.

4.9 Summary

Our work in this chapter presents a framework and methodology for evaluating a variety of thermal control options. Through simulation of architectural and thermal models we have examined a range of thermal management options. We have characterized all twelve policy combinations both in terms of instruction throughput and effective duty cycle.

Our best performing thermal control combination includes both control-theoretic distributed DVFS and a sensor-based migration policy. This design represents an elegant two-loop system allowing the migration policy to utilize feedback information from the core controllers. It also demonstrates the value of hardware-software collaboration on the thermal problem. Hardware performs fine-grained adjustments and ensures that thermal emergencies are avoided, while software uses migration to perform heat balancing and seeks to optimize the workload's performance.

Taken together, these studies create an overall picture regarding the issues and benefits of various thermal control policies and their combinations. DVFS mechanisms require added on-chip flexibility in the PLL and voltage modulation, but we show them to be robust and effective. Our migration schemes are fairly lightweight in implementation; they are designed to operate either with hardware performance counters (available on essentially all current processors) or feedback-based core control.

Although we have examined our spectrum of thermal control policies as a mixture among three axes, these are not the only possible dimensions. SMT and asymmetric cores are two possible extensions. Another shortcoming of our algorithm is that it is applicable mainly when interaction between the various threads is negligible. This assumption certainly does not hold true in the case of running parallel applications, which are becoming increasingly common nowadays. Future research will be needed to explore these issues.

Given the increasing challenges of thermal design in current and future processors, creative combinations of effective DTM policies are likely to be the only way to truly gain leverage on the problem. With that in mind, this chapter has offered a taxonomy of DTM techniques and has used the taxonomy to propose and explore interesting and novel DTM methods spanning from OS software down to control-theoretic hardware.

Chapter 5

Conclusions

5.1 Concluding Remarks

This thesis explores techniques for power and thermal management in multithreaded and multicore processors. The various contributions include novel concepts, methods, and simulation frameworks for power and temperature control in current and future microprocessor designs.

I have presented several experiments dealing with power management and dynamic thermal management. A range of issues to deal with include the power effect of scaling parallel applications, parameter variations, thermal control for multithreading, and migration in multicore processors. Through thermal management with simultaneous multithreading, I showed that the different power characteristics of different applications can be exploited in unison. For multicore processors, there is added spatial flexibility which can be exploited through distributed policies and migration. These concepts culminate in the holistic policy known as sensor-based migration, which migrates processes according to decisions based on thermal properties measured directly at runtime.

This thesis forms a comprehensive study of power and thermal management for

multithreaded and multicore processors. Through various techniques, I have shown that different environments, e.g. multithreaded vs multiprogrammed or SMT vs CMP, may each require a different set of practical management policies. The successful combined policies demonstrate that there is no single silver bullet for power and thermal management. Among the case studies for various scenarios, in this thesis I have provided:

- A model for estimating the trends in how parallel applications are affected by power variation, and experiments showing how different applications match or deviate from this model.
- A hardware-based thermal-control policy for simultaneous multithreading, showing that performance imbalance may be exploited for thermal management even at the single-core level.
- A fairly exhaustive taxonomy and corresponding quantitative study of multicore thermal management, providing a holistic view of both basic techniques and hybrid techniques.

As exemplified by the sensor-based migration with distributed voltage scaling, practical solutions for real processors will likely involve a range of techniques acting together.

5.2 Future Directions

Although the topics chosen for this thesis were carefully selected and focused on throughout these years, each course of study was picked among many other potential studies. Although I was not able to pursue every possible avenue, I hope that this work may serve as a solid foundation for others to follow on with future research.

As with any research prototype, the thermal management techniques demonstrated in this thesis may require much more development and testing before being

ready for actual product deployment. Although fairly exhaustive by including global and distributed policies for stop-go and DVFS in addition to migration policies, the experimental framework does not consider some other power-aware design techniques such as V_{DD} gating and heterogeneous cores. Similarly, the SMT-based adaptive thermal policy may be significantly more complex if integrated in conjunction with an ICOUNT [82] fetch policy. Furthermore, it is uncertain how any of the proposed algorithms can scale to much larger numbers of cores or more than two hotspots per core. There are numerous possibilities for conducting more exhaustive studies.

The multicore power management technique for parallel applications can be further extended and combined with other power management techniques. At this point, due to the natural interactions between threads in parallel applications, it is not a simple matter to apply a feedback control system as used in sensor-based migration for thermal management. Nonetheless, through further study it may be possible to formulate more holistic management techniques for parallel applications. These could be used in combination with other localized power management techniques such as DVFS or dynamic adaptive body biasing.

While the studies in this thesis have been limited to multiprogrammed environments or parallel applications using conventional thread-based shared-memory programming with locks and barriers, these are not the only possible programming models. Power and thermal management techniques could be devised for systems using transactional memory, thread-level speculation, or other parallel programming models that entail a high degree of speculative execution. These alternative platforms would likely entail different power and thermal management tradeoffs depending on the cost of speculative execution and the necessary interactions between threads. Just as moving from single-core to the multiple-core assumption dramatically revamped the thermal control taxonomy, it can be expected that different paradigms may open up further possibilities for holistic thermal management.

The successive chapters of this thesis have shown that refining each problem statement and its corresponding assumptions has lead to new methods and experimental outcomes. Similarly, incorporating new concepts—such as scaling these algorithms to larger numbers of threads and cores, implementing feedback-based thermal control for parallel applications, and speculative multithreading—may create many more possibilities for innovative thermal management.

Bibliography

[1] ACPI - Advanced Configuration and Power Interface. $http://www.acpi.info$, 2005.

[2] A. Agarwal, B. C. Paul, H. Mahmoodi, A. Datta, and K. Roy. A Process-Tolerant Cache Architecture for Improved Yield in Nanoscale Technologies. *IEEE Transactions on VLSI Systems*, 13(1), Jan. 2005.

[3] `amber(1)` manual page. BSD General Commands Manual, Dec. 2005.

[4] M. Annavaram, E. Grochowski, and J. Shen. Mitigating Amdahl's Law Through EPI Throttling. In *ISCA-32: Proc. of the 32nd Intl. Symp. on Computer Architecture*, June 2005.

[5] S. Balakrishnan, R. Rajwar, M. Upton, and K. Lai. The Impact of Performance Asymmetry in Emerging Multicore Architectures. In *ISCA-32: Proc. of the 32nd Intl. Symp. on Computer Architecture*, June 2005.

[6] S. Borkar. Design Challenges of Technology Scaling. *IEEE Micro*, pages 23–29, Jul/Aug. 1999.

[7] S. Borkar, T. Karnik, S. Narendra, J. Tschanz, A. Keshavarzi, and V. De. Parameter Variations and Impact on Circuits and Microarchitecture. In *DAC-40: Proc. of the 40th Design Automation Conf.*, June 2003.

[8] D. Brooks, P. Bose, S. Schuster, H. Jacobson, P. Kudva, A. Buyuktosunoglu, J.-D. Wellman, V. Zyuban, M. Gupta, and P. W. Cook. Power-Aware Microarchitecture: Design and Modeling Challenges for Next-Generation Microprocessors. *IEEE Micro*, 20(6):26–44, Nov/Dec. 2000.

[9] D. Brooks and M. Martonosi. Dynamic Thermal Management for High-Performance Microprocessors. In *HPCA-7: Proc. of the 7th Intl. Symp. on High-Performance Computer Architecture*, Jan. 2001.

[10] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: A Framework for Architectural-Level Power Analysis and Optimizations. In *ISCA-27: Proc. of the 27th Intl. Symp. on Computer Architecture*, June 2000.

[11] J. A. Butts and G. Sohi. A Static Power Model for Architects. In *MICRO-35: Proc. of the 33rd Intl. Symp. on Microarchitecture*, Dec. 2000.

[12] S. Chandra, K. Lahiri, A. Raghunathan, and S. Dey. Considering Process Variations During System-Level Power Analysis. In *ISLPED: Proc. of the Intl. Symp. on Low Power Electronics and Design*, Oct. 2006.

[13] P. Chaparro, J. González, and A. González. Thermal-Effective Clustered Microarchitectures. In *TACS: Proc. of the First Wkshp. on Temperature-Aware Computer Systems in Association with the 31st Intl. Symp. on Computer Architecture (ISCA-31)*, June 2004.

[14] P. Chaparro, J. González, G. Magklis, Q. Cai, and A. González. Understanding the Thermal Implications of Multi-Core Architectures. *IEEE Transactions on Parallel and Distributed Systems*, Aug. 2007. To appear.

[15] P. Chaparro, G. Magklis, J. González, and A. González. Distributing the Frontend for Temperature Reduction. In *HPCA-11: Proc. of the 11th Intl. Symp. on High-Performance Computer Architecture*, Feb. 2005.

[16] J. Clabes, J. Friedrich, M. Sweet, J. Dilullo, S. Chu, D. Plass, J. Dawson, P. Muench, L. Powell, M. Floyd, B. Sinharoy, M. Lee, M. Goulet, J. Wagoner, N. Schwartz, S. Runyon, G. Gorman, P. Restle, R. Kalla, J. McGill, and S. Dodson. Design and Implementation of the POWER5^{TM}Microprocessor. In *ISSCC '04: Proc. of the 2004 Intl. Solid-State Circuits Conf.*, Feb. 2004.

[17] CPU Maximum Operating Temperatures. *http : //www.gen − x − pc.com/cputemps.htm*. Gen-X PC, 2005.

[18] J. Donald and M. Martonosi. Temperature-Aware Design Issues for SMT and CMP Architectures. In *WCED-5: Proc. of the 5th Wkshp. on Complexity-Effective Design in Association with the 31st Intl. Symp. on Computer Architecture (ISCA-31)*, June 2004.

[19] J. Donald and M. Martonosi. Leveraging Simultaneous Multithreading for Adaptive Thermal Control. In *TACS-2: Proc. of the Second Wkshp. on Temperature-Aware Computer Systems in Association with the 32nd Intl. Symp. on Computer Architecture (ISCA-32)*, June 2005.

[20] J. Donald and M. Martonosi. An Efficient, Practical Parallelization Methodology for Multicore Architecture Simulation. *Computer Architecture Letters*, 5, Aug. 2006.

[21] J. Donald and M. Martonosi. Power Efficiency for Variation-Tolerant Multicore Processors: A Limits Study. In *ISLPED: Proc. of the Intl. Symp. on Low Power Electronics and Design*, Oct. 2006.

[22] J. Donald and M. Martonosi. Techniques for Multicore Thermal Management: Classification and New Exploration. In *ISCA-33: Proc. of the 32nd Intl. Symp. on Computer Architecture*, June 2006.

[23] M. Ekman and P. Stenström. Performance and Power Impact of Issue-width in Chip-Multiprocessor Cores. In *ICPP '03: Proc. of the 2003 Intl. Conf. on Parallel Processing*, Oct. 2003.

[24] W. El-Essawy and D. H. Albonesi. Mitigating Inductive Noise in SMT Processors. In *ISLPED: Proc. of the Intl. Symp. on Low Power Electronics and Design*, Aug. 2004.

[25] D. Ernst, N. S. Kim, S. Das, S. Pant, R. Rao, T. Pham, C. Ziesler, D. Blaauw, T. Austin, K. Flautner, and T. Mudge. Razor: A Low-Power Pipeline Based on Circuit-Level Timing Speculation. In *MICRO-36: Proc. of the 35th Intl. Symp. on Microarchitecture*, Dec. 2003.

[26] D. Genossar and N. S. Israel. Intel Pentium®M Processor Power Estimation, Budgeting, Optimization, and Validation. *Intel Technology Journal*, 7(2), May 2003.

[27] S. Ghiasi and D. Grunwald. Design Choices for Thermal Control in Dual-Core Processors. In *WCED-5: Proc. of the 5th Wkshp. on Complexity-Effective Design in Association with the 31st Intl. Symp. on Computer Architecture (ISCA-31)*, June 2004.

[28] A. González. Research Challenges on Temperature-Aware Computer Systems (panel). In *TACS-2: Second Wkshp. on Temperature-Aware Computer Systems*. Intel Corp., June 2005.

[29] S. Gunther, F. Binns, D. M. Carmean, and J. C. Hall. Managing the Impact of Increasing Microprocessor Power Consumption. *Intel Technology Journal*, Q1, 2001.

[30] J. L. Gustafson. Reevaluating Amdahl's Law. *Communications of the ACM*, 31(5):532–533, May 1988.

[31] Y. Han, I. Koren, and C. A. Moritz. Temperature Aware Floorplanning. In *TACS-2: Proc. of the Second Wkshp. on Temperature-Aware Computer Systems in Association with the 32nd Intl. Symp. on Computer Architecture (ISCA-32)*, June 2005.

[32] J. Hasan, A. Jalote, T. N. Vijaykumar, and C. Brodley. Heat Stroke: Power-Density-Based Denial of Service in SMT. In *HPCA-11: Proc. of the 11th Intl. Symp. on High-Performance Computer Architecture*, Feb. 2005.

[33] J. L. Henning. SPEC CPU2000: Measuring CPU Performance in the New Millennium. *IEEE Computer*, 33(7):28–35, July 2000.

[34] S. Heo, K. Barr, and K. Asanović. Reducing Power Density through Activity Migration. In *ISLPED: Proc. of the Intl. Symp. on Low Power Electronics and Design*, Aug. 2003.

[35] W. Huang, M. R. Stan, K. Skadron, K. Sankaranarayanan, S. Ghosh, and S. Velusamy. Compact Thermal Modeling for Temperature-Aware Design. In *DAC-41: Proc. of the 41st Design Automation Conf.*, June 2004.

[36] E. Humenay, D. Tarjan, and K. Skadron. Impact of Parameter Variations on Multi-Core Chips. In *ASGI: Proc. of the Wkshp. on Architectural Support for Gigascale Integration in Association with the 33rd Intl. Symp. on Computer Architecture (ISCA-33)*, June 2006.

[37] E. Humenay, D. Tarjan, and K. Skadron. Impact of Process Variations on Multi-core Performance Symmetry. In *DATE: Proc. of the Symp. on Design, Automation, and Test in Europe*, Apr. 2007.

[38] Hyper-Threading Technology. *http : //www.intel.com/technology/hyperthread/*. Intel Corporation, 2006.

[39] Intel Products Desktop Processor Roadmap. *http : //www.intel.com/products/roadmap/index.htm*, 2007.

[40] A. Iyer and D. Marculescu. Power Efficiency of Multiple Clock Multiple Voltage Cores. In *ICCAD: Proc. of the Intl. Conf. on Computer-Aided Design*, Nov. 2002.

[41] I. Kadayif, M. Kandemir, and U. Sezer. An Integer Linear Programming Based Approach for Parallelizing Applications in On-Chip Multiprocessors. In *DAC-39: Proc. of the 39th Design Automation Conf.*, June 2002.

[42] S. Kaxiras, G. Narlikar, A. D. Berenbaum, and Z. Hu. Comparing Power Consumption of an SMT and a CMP DSP for Mobile Phone Workloads. In *CASES '01: Proc. of the 2001 Intl. Conf. on Compilers, Architecture, and Synthesis for Embedded Systems*, Nov. 2001.

[43] A. Kumar, L. Shang, L.-S. Peh, and N. K. Jha. HybDTM: A Coordinated Hardware-Software Approach for Dynamic Thermal Management. In *DAC-43: Proc. of the 43rd Design Automation Conf.*, July 2006.

[44] R. Kumar, K. Farkas, N. Jouppi, P. Ranganathan, and D. Tullsen. Single-ISA Heterogeneous Multi-Core Architectures: The Potential for Processor Power Reduction. In *MICRO-36: Proc. of the 35th Intl. Symp. on Microarchitecture*, Dec. 2003.

[45] J. Laudon. Performance/Watt: The New Server Focus. In *dasCMP: Proc. of the Wkshp. on Design, Analysis, and Simulation of Chip Multiprocessors in Association with the 38th Intl. Symp. on Microarchitecture (MICRO-38)*, Nov. 2005.

[46] K.-J. Lee and K. Skadron. Using Performance Counters for Runtime Temperature Sensing in High-Performance Processors. In *HP-PAC: Proc. of the Wkshp. on High-Performance, Power-Aware Computing*, Apr. 2005.

[47] J. Li and J. F. Martínez. Power-Performance Implications of Thread-level Parallelism on Chip Multiprocessors. In *$P = AC^2$: IBM Conf. on Architecture, Circuits, and Compilers*, Oct. 2004.

[48] J. Li and J. F. Martínez. Power-performance implications of thread-level parallelism on chip multiprocessors. In *ISPASS: Proc. of the Intl. Symp. on Performance Analysis of Systems and Software*, Mar. 2005.

[49] J. Li and J. F. Martínez. Dynamic Power-Performance Adaptation of Parallel Computation on Chip Multiprocessors. In *HPCA-12: Proc. of the 12th Intl. Symp. on High-Performance Computer Architecture*, Feb. 2006.

[50] K. Li and P. Hudak. Memory Coherence in Shared Virtual Memory Systems. *ACM Transactions on Computer Systems*, 7(4), Nov. 1989.

[51] Y. Li, D. Brooks, Z. Hu, and K. Skadron. Performance, Energy, and Thermal Considerations for SMT and CMP Architectures. In *HPCA-11: Proc. of the 11th Intl. Symp. on High-Performance Computer Architecture*, Feb. 2005.

[52] Y. Li, D. Brooks, Z. Hu, K. Skadron, and P. Bose. Understanding the Energy Efficiency of Simultaneous Multithreading. In *ISLPED: Proc. of the Intl. Symp. on Low Power Electronics and Design*, Aug. 2004.

[53] Y. Li, B. Lee, D. Brooks, Z. Hu, and K. Skadron. CMP Design Space Exploration Subject to Physical Constraints. In *HPCA-12: Proc. of the 12th Intl. Symp. on High-Performance Computer Architecture*, Feb. 2006.

[54] X. Liang and D. Brooks. Latency Adaptation of Multiported Register Files to Mitigate Variations. In *ASGI: Proc. of the Wkshp. on Architectural Support for Gigascale Integration in Association with the 33rd Intl. Symp. on Computer Architecture (ISCA-33)*, June 2006.

[55] X. Liang and D. Brooks. Mitigating the Impact of Process Variations on CPU Register File and Execution Units. In *MICRO-38: Proc. of the 39th Intl. Symp. on Microarchitecture*, June 2006.

[56] Massively Parallel Technologies. *http : //www.massivelyparallel.com/*, 2006.

[57] J. McGregor. x86 Power and Thermal Management. *Microprocessor Report*, Dec. 2004.

[58] K. Meng and R. Joseph. Process Variation Aware Cache Leakage Management. In *ISLPED: Proc. of the Intl. Symp. on Low Power Electronics and Design*, Oct. 2006.

[59] A. Merkel, A. Weissel, and F. Bellosa. Event-Driven Thermal Management in SMP Systems. In *TACS-2: Proc. of the Second Wkshp. on Temperature-Aware Computer Systems in Association with the 32nd Intl. Symp. on Computer Architecture (ISCA-32)*, June 2005.

[60] M. Moudgill, J.-D. Wellman, and J. H. Moreno. Environment for PowerPC Microarchitecture Exploration. *IEEE Micro*, 19(3):15–25, May/June 1999.

[61] S. Naffziger. Dynamically Optimized Power Efficiency with Foxton Technology. In *Proc. of Hot Chips 17*, Aug. 2005.

[62] M. S. Papamarcos and J. H. Patel. A Low-Overhead Coherence Solution for Multiprocessors with Private Cache Memories. In *ISCA-11: Proc. of the 11th Intl. Symp. on Computer Architecture*, June 1984.

[63] C. D. Patel. Smart Chip, System and Data Center: Dynamic Provisioning of Power and Cooling from Chip Core to the Cooling Tower (keynote). In *TACS-2: Second Wkshp. on Temperature-Aware Computer Systems*. HP Labs, June 2005.

[64] E. Perelman, G. Hamerly, and B. Calder. Picking Statistically Valid and Early Simulation Points. In *PACT '03: Proc. of the 12th Intl. Conf. on Parallel Architectures and Compilation Techniques*, Sept. 2003.

[65] F. Pollack. New Microarchitecture Challenges in the Coming Generations of CMOS Process Technologies. Keynote presentation for *MICRO-32: 32nd Intl. Symp. on Microarchitecture*. Intel Corp., Nov. 1999.

[66] M. D. Powell, M. Gomaa, and T. N. Vijaykumar. Heat-and-Run: Leveraging SMT and CMP to Manage Power Density Through the Operating System. In *ASPLOS-XI: Proc. of the 11th Intl. Conf. on Architectural Support for Programming Languages and Operating Systems*, Oct. 2004.

[67] R. Sasanka, S. V. Adve, Y.-K. Chen, and E. Debes. The Energy Efficiency of CMP vs. SMT for Multimedia Workloads. In *ICS '04: Proc. of the 18th Intl. Conf. on Supercomputing*, June 2004.

[68] J. Seng, D. Tullsen, and G. Cai. Power-Sensitive Multithreaded Architecture. In *ICCD: Proc. of the Intl. Conf. on Computer Design*, Sept. 2000.

[69] L. Shang, L.-S. Peh, A. Kumar, and N. K. Jha. Thermal Modeling, Characterization and Management of On-Chip Networks. In *MICRO-37: Proc. of the 37th Intl. Symp. on Microarchitecture*, Dec. 2004.

[70] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder. Automatically Characterizing Large Scale Program Behavior. In *ASPLOS-X: Proc. of the 10th Intl. Conf. on Architectural Support for Programming Languages and Operating Systems*, Oct. 2002.

[71] Y. Shi. Reevaluating Amdahl's Law and Gustafson's Law. In *Temple University Technical Report*. Computer Sciences Department (MS:38-24), Oct. 1996.

[72] K. Skadron. Thermal Issues for Temperature-Aware Computer Systems (tutorial). In *ISCA-31: 31st Intl. Symp. on Computer Architecture*, June 2004.

[73] K. Skadron, T. Abdelzaher, and M. R. Stan. Control-Theoretic Techniques and Thermal-RC Modeling for Accurate and Localized Dynamic Thermal Management. In *HPCA-8: Proc. of the 8th Intl. Symp. on High-Performance Computer Architecture*, Feb. 2002.

[74] K. Skadron, M. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan. Temperature-Aware Microarchitecture. In *ISCA-30: Proc. of the 30th Intl. Symp. on Computer Architecture*, Apr. 2003.

[75] A. Snavely and D. Tullsen. Symbiotic Jobscheduling for a Simultaneous Multithreading Processor, Nov. 2000.

[76] J. Srinivasan, S. V. Adve, P. Bose, and J. A. Rivers. The Case for Lifetime Reliability-Aware Microprocessors. In *ISCA-31: Proc. of the 31st Intl. Symp. on Computer Architecture*, June 2004.

[77] D. C. Steere, A. Goel, J. Gruenburg, D. McNamee, C. Pu, and J. Walpole. A Feedback-driven Proportion Allocator for Real-rate Scheduling. In *OSDI '99: Proc. of the Third Symp. on Operating System Design and Implementation*, Feb. 1999.

[78] V. Tiwari, D. Singh, S. Rajgopal, G. Mehta, R. Patel, and F. Baez. Reducing Power in High-Performance Microprocessors. In *DAC-35: Proc. of the 35th Design Automation Conf.*, June 1998.

[79] M. Tremblay. High Performance Throughput Computing (Niagara). Keynote presentation for *31st ISCA-31: 31st Intl. Symp. on Computer Architecture*. Sun Microsystems, June 2004.

[80] J. Tschanz, K. Bowman, and V. De. Variation-Tolerant Circuits: Circuit Solutions and Techniques. In *DAC-42: Proc. of the 42nd Design Automation Conf.*, June 2005.

[81] D. Tullsen and J. Brown. Handling Long-Latency Loads in a Simultaneous Multithreaded Processor. In *MICRO-35: Proc. of the 34th Intl. Symp. on Microarchitecture*, Nov. 2001.

[82] D. Tullsen, S. Eggers, and H. Levy. Simultaneous Multithreading: Maximizing On-Chip Parallelism. In *ISCA-22: Proc. of the 22nd Intl. Symp. on Computer Architecture*, June 1995.

[83] X. Vera, O. Unsal, and A. González. X-Pipe: An Adaptive Resilient Microarchitecture for Parameter Variations. In *ASGI: Proc. of the Wkshp. on Architectural Support for Gigascale Integration in Association with the 33rd Intl. Symp. on Computer Architecture (ISCA-33)*, June 2006.

[84] A. Weissel and F. Bellosa. Dynamic Thermal Management for Distributed Systems. In *TACS: Proc. of the First Wkshp. on Temperature-Aware Computer Systems in Association with the 31st Intl. Symp. on Computer Architecture (ISCA-31)*, June 2004.

[85] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Goopta. The SPLASH-2 programs: Characterization and methodological considerations. In *ISCA-22: Proc. of the 22nd Intl. Symp. on Computer Architecture*, June 1995.

[86] Q. Wu, P. Juang, M. Martonosi, and D. W. Clark. Formal Online Methods for Voltage/Frequency Control in Multiple Clock Domain Microprocessors. In *ASPLOS-XI: Proc. of the 11th Intl. Conf. on Architectural Support for Programming Languages and Operating Systems*, Oct. 2004.

[87] Q. Wu, P. Juang, M. Martonosi, and D. W. Clark. Voltage and Frequency Control with Adaptive Reaction Time in Multiple-Clock-Domain Processors. In *HPCA-11: Proc. of the 11th Intl. Symp. on High-Performance Computer Architecture*, Feb. 2005.

[88] Q. Wu, V. Reddi, Y. Wu, J. Lee, D. Connors, D. Brooks, M. Martonosi, and D. W. Clark. A Dynamic Compilation Framework for Controlling Microprocessor Energy and Performance. In *MICRO-38: Proc. of the 38th Intl. Symp. on Microarchitecture*, Nov. 2005.