UNIVERSITY OF CALIFORNIA, SAN DIEGO

Efficient Thermal Management for Multiprocessor Systems

A dissertation submitted in partial satisfaction of the
requirements for the degree
Doctor of Philosophy

in

Computer Science and Engineering

by

Ayşe Kıvılcım Coşkun

Committee in charge:

Tajana Šimunić Rosing, Chair
Kenny C. Gross
Rajesh Gupta
Tara Javidi
Andrew B. Kahng
Dean Tullsen

2009

The dissertation of Ayşe Kıvılcım Coşkun is approved, and it is acceptable in quality and form for publication on microfilm and electronically:

_____

_____

_____

_____

_____

_____
Chair

University of California, San Diego

2009

DEDICATION

To my parents and my sister.

EPIGRAPH

For a successful technology,
reality must take precedence over public relations,
for nature cannot be fooled.
—Richard P. Feynman

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

I would like to thank my advisor, Dr. Tajana Simunic Rosing, for her guidance during my PhD. I must also thank my doctoral committee, Dr. Kenny Gross, Dr. Rajesh Gupta, Dr. Tara Javidi, Dr. Andrew B. Kahng, and Dr. Dean Tullsen for their valuable feedback and contributions.

My internship at Sun Microsystems during my PhD has been an invaluable experience, and I appreciate the help, support, and the enjoyable working environment my colleagues at Sun have provided. I especially thank the System Dynamics Characterization and Control team at Sun, including Dr. Keith Whisnant. I am also grateful to Dr. David Atienza and Dr. Giovanni De Micheli, as it has been a pleasure to work with them during the summers I spent at EPFL and afterwards.

I cannot thank Dr. Yusuf Leblebici enough for his contributions to my academic career since my early years in college, and for his insightful advice during critical times. I also thank Dr. Brad Calder for his guidance when I was trying to find my research direction.

I would like to thank all my lab mates, co-authors, and friends at UC San Diego, especially Gaurav Dhiman, Shervin Sharifi, Giacomo Marchetti, Edoardo Regini, Priti Aghera, Raid Ayoub, Richard Strong, Kresimir Mihic, and the Architecture Lab folks for their friendship and contributions to my research during my PhD years. I thank also my professors at Sabanci University, Turkey, for inspiring me to pursue an academic career.

I have been surrounded with an exceptional circle of friends, and would like to thank especially Ozge Cavus, Fikriye Kurban, Selim Guncer, Didem Turker, Hasim Mardin, Vuslat and Beste Nazilli, Deniz Kebabci, and Justin Burke for their support, tolerance, and for sharing the ups and downs. I thank Denis Dondi for his love and companionship, and for his patience during stressful times.

The greatest thanks are for all of my family, who had my best interest in their hearts at all times, who loved, supported and motivated me. I feel extremely lucky to have such a big and wonderful family. I am dedicating this thesis to my parents, Nermin Sungar and Umur Coskun, and my sister, Dr. Safak Coskun, for they have always encouraged my curiosity and my ambition to follow my dreams. They have made me the person I am today, taught me to believe in myself, and reminded me what is important in life.

The research that forms the basis of this dissertation has been funded in part by Sun Microsystems, UC MICRO, Center for Networked Systems (CNS) at UCSD, Cisco, Qualcomm, IBM, MARCO/DARPA Gigascale Systems Research Center, NSF MRI Greenlight #0821155, and NSF CCF #0916127.

The text of Chapter 2 is in part a reprint of the material from the paper, *Ayse K. Coskun, Richard Strong, Dean Tullsen and Tajana Rosing, "Evaluating the Impact of Job Scheduling and Power Management on Processor Lifetime for Chip Multiprocessors"*, in Proceedings of SIGMETRICS/Performance 2009–Joint International Conference on Measurement and Modeling of Computer Systems, 2009. The dissertation author was the primary researcher and author, and the co-authors involved in the publication [22] directed, supervised, and assisted in the research which forms the basis for that material.

The text of Chapter 3 is in part a reprint of the material from the paper, *Ayse K. Coskun, Tajana Simunic Rosing, Keith Whisnant and Kenny Gross, "Static and Dynamic Temperature-Aware Scheduling for Multiprocessor SoCs"*, in IEEE Transactions on VLSI, September 2008. The dissertation author was the primary researcher and author, and the co-authors involved in the publication [21] directed, supervised, and assisted in the research which forms the basis for that material.

The text of Sections 4.1 and 4.3 are in part reprints of the material from the papers, *Ayse K. Coskun, Tajana Simunic Rosing and Keith Whisnant, "Temperature Aware Task Scheduling in MPSoCs"*, in Proceedings of Design Automation and Test in Europe (DATE), 2007, and *Ayse K. Coskun, Tajana Simunic Rosing, Keith Whisnant and Kenny Gross, "Static and Dynamic Temperature-Aware Scheduling for Multiprocessor SoCs"*, in IEEE Transactions on VLSI, September 2008. The dissertation author was the primary researcher and author, and the co-authors involved in the publications [17] and [21] directed, supervised, and assisted in the research which forms the basis for that material.

The text of Sections 4.2 and 4.3 are in part a reprint of the material from the paper, *Ayse K. Coskun, Tajana Simunic Rosing and Kenny Gross, "Temperature Management in Multiprocessor SoCs Using Online Learning"*, in Proceedings of Design Automation Conference (DAC), 2008. The dissertation author was the primary researcher and author, and the co-authors involved in the publication [19] directed, supervised, and assisted in the research which forms the basis for that material.

The text of Chapter 5 is in part a reprint of the material from the papers, *Ayse*

K. Coskun, Tajana Simunic Rosing, and Kenny Gross, "Utilizing Predictors for Efficient Thermal Management in Multiprocessor SoCs", in IEEE Transactions on CAD, 2009, and *Ayse K. Coskun, Tajana Simunic Rosing and Kenny Gross, "Proactive Temperature Balancing for Low Cost Thermal Management in MPSoCs"*, in Proceedings of International Conference on Computer-Aided Design (ICCAD), 2008. The dissertation author was the primary researcher and author, and the co-authors involved in the publications [20] and [15] directed, supervised, and assisted in the research which forms the basis for that material.

The text of Section 6.3.1 is in part a reprint of the material from the paper, *Ayse K. Coskun, Tajana Simunic Rosing, Jose Ayala, David Atienza and Yusuf Leblebici, "Dynamic Thermal Management in 3D Multicore Architectures"*, in Proceedings of Design Automation and Test in Europe (DATE), 2009. The dissertation author was the primary researcher and author, and the co-authors involved in the publication [18] directed, supervised, and assisted in the research which forms the basis for that material.

The text of Section 6.3.1 is in part a reprint of the material from the paper, *Ayse K. Coskun, Andrew B. Kahng and Tajana Simunic Rosing, "Temperature- and Cost-Aware Design of 3D Multiprocessor Architectures"*, in Proceedings of Euromicro Conference on Digital System Design (DSD), 2009. The dissertation author was the primary researcher and author, and the co-authors involved in the publication [16] directed, supervised, and assisted in the research which forms the basis for that material.

VITA

2003	B. S. in Microelectronics Engineering, Sabanci University, Istanbul, Turkey
2003-2009	Graduate Student Researcher, University of California, San Diego
2006	M. S. in Computer Science and Engineering, University of California, San Diego
2006-2009	Intern in System Dynamics Characterization and Control (SDCC), Sun Microsystems, San Diego
2009	Ph. D. in Computer Science and Engineering, University of California, San Diego

PUBLICATIONS

Ayse K. Coskun, Tajana Simunic Rosing, and Kenny Gross. "Utilizing Predictors for Efficient Thermal Management in Multiprocessor SoCs". *IEEE Transactions on CAD*, To appear in 2009.

Ayse K. Coskun, Tajana Simunic Rosing, Keith Whisnant and Kenny Gross. "Static and Dynamic Temperature-Aware Scheduling for Multiprocessor SoCs". *IEEE Transactions on VLSI*, vol.16 no.9, pp. 1127-1140, Sept. 2008.

Ayse K. Coskun, Tajana Simunic Rosing, Kresimir Mihic, Yusuf Leblebici and Giovanni De Micheli. "Analysis and Optimization of MPSoC Reliability". *Journal of Low Power Electronics (JOLPE)*, vol.2 no.1, pp.56-69, April 2006.

Ayse K. Coskun, Tajana Simunic Rosing, Jose Ayala and David Atienza. "Modeling and Dynamic Management of 3D Multicore Systems with Liquid Cooling". To appear in *IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC)*, October 2009.

Francesco Zanini, Ayse K. Coskun, David Atienza and Giovanni De Micheli. "Optimal Multi-Processor SoC Thermal Simulation via Adaptive Differential Equation Solvers". To appear in *IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC)*, October 2009.

Ayse K. Coskun, Tajana Rosing and Andrew B. Kahng. "Temperature- and Cost-Aware Design of 3D Multiprocessor Architectures". *Euromicro Conference on Digital System Design (DSD)*, pp. 183-190, 2009.

Ayse K. Coskun, Richard Strong, Dean Tullsen and Tajana Rosing. "Evaluating the Impact of Job Scheduling and Power Management on Processor Lifetime for Chip Multiprocessors". *SIGMETRICS/Performance 2009–Joint International Conference on Measurement and Modeling of Computer Systems*, pp.169-180, 2009.

Ayse K. Coskun, Tajana Simunic Rosing, Jose Ayala, David Atienza and Yusuf Leblebici. "Dynamic Thermal Management in 3D Multicore Architectures". *Proceedings of Design Automation and Test in Europe (DATE)*, pp. 1410-1415, 2009.

Ayse K. Coskun, Tajana Simunic Rosing and Kenny Gross. "Proactive Temperature Balancing for Low Cost Thermal Management in MPSoCs". *Proceedings of International Conference on Computer-Aided Design (ICCAD)*, pp. 250-257, 2008.

Ayse K. Coskun, Tajana Simunic Rosing and Kenny Gross. "Proactive Temperature Management in MPSoCs". *Proceedings of International Symposium on Low Power Electronics and Design (ISLPED)*, pp. 165-170, 2008.

Ayse K. Coskun, Tajana Simunic Rosing and Kenny Gross. "Temperature Management in Multiprocessor SoCs Using Online Learning". *Proceedings of Design Automation Conference (DAC)*, pp. 890-893, 2008.

Ayse K. Coskun, Tajana Simunic Rosing, Keith Whisnant and Kenny Gross. "Temperature-Aware MPSoC Scheduling for Reducing Hot Spots and Gradients". *Proceedings of Asia and South Pacific Design Automation Conference (ASPDAC)*, pp. 49-54, 2008.

Ayse K. Coskun, Tajana Simunic Rosing and Keith Whisnant. "Temperature Aware Task Scheduling in MPSoCs". *Proceedings of Design Automation and Test in Europe (DATE)*, pp. 1659-1664, 2007.

Satish Narayanasamy, Ayse K. Coskun and Brad Calder. "Transient Fault Prediction Based on Anomalies in Processor Events". *Proceedings of Design Automation and Test in Europe (DATE)*, pp. 1140-1145, 2007.

Ayse K. Coskun, Tajana Simunic Rosing, Yusuf Leblebici and Giovanni De Micheli. "A Simulation Methodology for Reliability Analysis in Multi-Core SoCs". *Great Lakes Symposium on VLSI (GLSVLSI)*, pp. 95-99, 2006.

ABSTRACT OF THE DISSERTATION

Efficient Thermal Management for Multiprocessor Systems

by

Ayşe Kıvılcım Coşkun
Doctor of Philosophy in Computer Science and Engineering
University of California San Diego, 2009
Tajana Šimunić Rosing, Chair

High temperatures and large thermal variations on the die create severe challenges in system reliability, performance, leakage power, and cooling costs. Designing for worst-case thermal conditions is highly costly and time-consuming. Therefore, dynamic thermal management methods are needed to maintain safe temperature levels during execution. Conventional management techniques sacrifice performance to control temperature and only consider the hot spots, neglecting the effects of thermal variations. This thesis focuses on developing performance-efficient techniques to achieve safe and balanced thermal profiles on multiprocessor system-on-chips (MPSoCs).

Modeling performance, temperature, and reliability of MPSoCs with high accuracy and reasonable simulation time is a challenge, because we need to keep track of instruction-level activities and also simulate sufficiently long real-time execution to have meaningful reliability estimates. The first contribution of this thesis is a fast simulation framework, which evaluates reliability of runtime policies or design-time decisions accurately in a matter of hours—whereas traditional architecture-level simulators would have to run for days.

Job scheduling on an MPSoC has a significant impact on temperature and reliability. For systems with *a priori* known workloads, this thesis proposes a scheduling optimization method which outperforms other static energy or temperature management techniques in terms of reducing thermal hot spots and gradients. However, having an accurate design-time workload estimate is not possible for most systems. This work introduces dynamic techniques to address runtime variations in workload. The key aspects of these dynamic techniques are low-performance impact and adaptation capability.

Reacting after thermal events occur reduces the efficiency of thermal management policies. This thesis proposes a novel proactive management approach to address this issue, and shows that utilizing a thermal forecast for temperature-aware scheduling achieves significant gains in both temperature and performance. All the novel management policies introduced in this thesis are evaluated using an experimental framework based on real-life systems and workloads. In the experiments on an UltraSPARC T1 processor, proactive thermal management achieves remarkable results with an average 60% reduction in hot spot occurrences, 80% reduction in spatial gradients and 75% reduction in thermal cycles in comparison to reactive thermal management, while also improving performance.

Chapter 1

Introduction

Power consumption of chips has increased substantially for the last few decades to accommodate the rising performance demand. Higher power consumption combined with smaller device dimensions increase power densities and on-chip temperatures. A number of critical challenges arise due to high temperatures, including lower reliability, higher cooling costs, and performance degradation. The difficulty of managing these challenges, especially the prohibitively high cooling costs, have motivated the shift from designing single-core highly-complex architectures to multicore chips that integrate several simpler, lower power cores [38]. Today, advances in process technology enable manufacturing complete multiprocessor SoCs including CPUs, memories and communication architectures on a single die. The Sun UltraSPARC T1 [38] and the IBM Cell [35] are examples of such multicore processors.

The hardware parallelism supplied by the multicore architectures enables achieving higher performance per Watt. However, the power density is still at a rising trend, as we continue to shrink the feature sizes and improve the performance of our systems at the cost of higher power consumption. In deep submicron process technologies, high temperatures, process imperfections, and reduced voltage margins have already made the systems much more vulnerable to failures. In addition, as we progress to designing many-core systems, we manufacture larger chips which potentially have dramatically higher temperature variations across the die. These variations add to the existing challenges caused by high temperatures, as they degrade system reliability, performance, and cooling efficiency. Therefore, multicore systems still face a great amount of temperature-induced problems.

To address the need of efficient thermal management of multicore systems, this thesis analyzes the effects of design-time and runtime decisions on temperature, and develops techniques to manage temperature without a substantial effect on performance. This introductory chapter provides an overview of the temperature-induced challenges and the prior work in thermal management. It also highlights the contributions of the thesis.

1.1 Temperature-Induced Challenges

One of the obvious results of high temperatures is higher cooling costs. This increase is due to both the need for more expensive packaging solutions and to the cooling energy consumed by the system fans that are supplying the air to cool the package. Leakage power, which is a significant portion of total power for $65nm$ and below, is exponentially related to temperature. High temperatures adversely affect performance as well, as the effective operating speed of devices decreases with high temperatures.

Temperature also has a strong effect on system reliability. Hot spots exponentially accelerate failure mechanisms such as electromigration, stress migration, and dielectric breakdown, which cause permanent device failures [34]. In fact, a small difference in the operating temperature (i.e., $10-15^oC$) can result in a 2X difference in the lifetime of the devices [70].

Addressing thermal hot spots alone is not enough to improve reliability. Temporal and spatial thermal gradients affect device reliability even at moderate temperatures [44]. The failure rate due to thermal cycling (i.e., temporal fluctuations in temperature) increases with higher magnitude and frequency of the temperature cycles [34]. For example, a 10^oC increase in the magnitude of cycles can cause about 16X decrease in mean-time-to-failure of metallic structures [34]. Thermal cycling causes accelerated package fatigue and plastic deformations of materials, leading to permanent failures. Such cycles are created either by low frequency power changes such as system power on/off cycles, or by more frequent workload rate changes and power management decisions (e.g., putting idle units in deep sleep mode) [52].

High spatial temperature gradients, which can easily occur on today's large multicore SoCs, also cause performance and reliability degradation. In process technologies below 0.13 μm, reliability issues arise due negative bias temperature instability (NBTI) and hot carrier injection (HCI) effects, as the operating temperatures and electric fields

reach high enough values to accelerate these mechanisms during device lifetime [39]. When the temperature gradient is large, the delay characteristics of devices may change in a sufficiently different amount due to NBTI and HCI to cause the circuits to fail in meeting timing constraints [39]. In addition, increasing temperature increases local resistances, and as a result, elevates circuit delays and IR drop as well [55]. Global clock networks are especially vulnerable to large spatial variations. Every 20 degrees increase in temperature causes 5-6% increase in Elmore delay in interconnects. As a result, clock skew problems become noticeable for spatial variations of even 15-20 degrees [1]. Another important adverse effect of spatial gradients is lower cooling efficiency. As the cooling infrastructure (i.e., heat sink and fans) needs to take care of the highest temperature on the chip, large on-chip variations result in over-cooling and waste energy.

Initially, designers have addressed thermal challenges at design time to ensure that the reliability, performance, and leakage power constraints are met. Temperature-aware floorplanning (e.g., [54]) or improving the cooling infrastructure are examples of offline techniques. However, due to the increase in power densities with each new technology node and the high integration levels in systems, designing for worst-case has become very costly and time consuming [11]. Therefore, designing for better-than-worst-case conditions and implementing dynamic thermal management strategies to guarantee safe temperatures at runtime have become a common practice.

Conventional dynamic thermal management methods maintain temperature below critical values by slowing down or stalling the processor upon reaching a predetermined threshold temperature value (e.g., [59])—hence, such methods trade-off performance for reliability. Well-known techniques for thermal management include fetch-toggling [59], dynamic voltage/frequency scaling [59, 24], thread migration [29] or hybrid techniques that combine two or more of management methods to improve performance [40].

1.2 Thesis Contributions

Prior thermal management work has focused on thermal hot spots, but the effects of thermal variations in time and space have not been considered. For example, a dynamic power management (DPM) policy that turns off idle cores to save energy can have conflicting goals with a thermal management policy that lowers and balances the temperature. DPM without thermal constraints can create large cycles or gradi-

ents due to the ultra-low power sleep modes it utilizes, and consequently can degrade reliability—even though it may reduce the average temperature at the same time. Therefore, developing policies to take care of both hot spots and variations while maintaining desired performance and energy levels is an important challenge. This thesis focuses on the design of performance-efficient thermal management techniques that prevent the temperature-induced reliability and performance degradation as much as possible, as opposed to only avoiding critically high temperatures.

Achieving performance-efficiency requires identifying the workload characteristics at design time or at runtime and exploiting these characteristics to make intelligent management decisions. For MPSoCs, workload scheduling has a substantial effect on temperature. Thus, temperature-aware scheduling has the potential to overcome temperature-induced challenges at a low performance cost. To this end, for systems with known workloads (such as some embedded systems), this work introduces a job scheduling optimization method that minimizes hot spots and gradients while maintaining the timing requirements of the workload.

In most real-life systems, it is difficult to accurately estimate runtime workload conditions during design. Workload varies at runtime, requiring dynamic management methods to manage temperature. Multiprocessor systems provide several advantages for dynamic management. First, MPSoCs are typically under-utilized for a significant portion of their lifetime. Therefore, intelligent workload allocation and scheduling policies can improve the thermal profile dynamically at a low performance cost in comparison to slowing down or stalling execution. Second, many chips today contain a number of sensors and counters that provide real-time information about the system dynamics (e.g., temperature, performance, etc.). The thermal management policies proposed in this work utilize such information for tuning the runtime management policy to efficiently fit the current workload conditions. We utilize these properties of MPSoCs to design low-cost dynamic scheduling techniques that reduce and balance temperature.

Existing dynamic thermal management methods are reactive, that is, they take action after a thermal emergency occurs. This work discusses a novel technique to forecast temperature dynamics, and shows how this forecast can be utilized for proactive management. Proactive temperature-aware job scheduling tracks the system dynamics as collected by the sensors, learns these dynamics, estimates the near-future temperature, and adjusts job scheduling in advance to mitigate thermal problems before they occur.

This way, the system can operate at a much more desirable performance-temperature trade-off point in comparison to reacting to thermal emergencies.

The contributions of this thesis are outlined as the following:

- It provides a simulation framework for analyzing the effects of runtime management decisions (e.g., workload migration, power management, voltage/frequency scaling, job scheduling) in multiprocessor systems with high accuracy and reasonable simulation time. The details of this simulation framework are in Chapter 2.

- For systems with a known set of jobs, it proposes an optimization technique to compute the static schedule for minimizing the hot spots and thermal gradients. The optimization technique is able to reduce the frequency of hot spots by 35%, spatial gradients by 85% and thermal cycles by 61% in comparison to the optimal schedule for minimizing energy. This static scheduling method is discussed in Chapter 3.

- As workload varies during execution for many real-life systems, this thesis introduces two dynamic management techniques: (1) *Adaptive-Random*, which is a temperature-aware scheduling policy that reduces the hot spots and thermal gradients; and (2) *Online Learning*, which selects the best policy for the current workload among a given set of thermal management and scheduling policies. Both techniques have very-low runtime overhead and adaptation capability. Chapter 4 provides the details of these low-overhead dynamic management techniques.

- It shows that reactive thermal management strategies cannot effectively utilize the performance capacity of multicore architectures, and proposes a proactive method to learn the system dynamics and forecast thermal events before they occur. Using the thermal forecast, *Proactive Temperature Balancing* achieves a significantly better temperature-performance trade-off in comparison to reactive thread migration or voltage/frequency scaling methods. The proactive management method is discussed in Chapter 5.

- The experimental methodology to analyze and evaluate the proposed methods is based on real-life systems and workloads. The dynamic job scheduling methods are implemented on an UltraSPARC T1 processor [38]. In the experiments performed on the UltraSPARC T1, proactive thermal management achieves 60% reduction

in hot spot occurrences, 80% reduction in spatial gradients and 75% reduction in thermal cycles on average in comparison to reactive thermal management, while also improving performance.

Chapter 2

Temperature and Reliability Simulation

Modeling temperature and the effects of temperature on reliability is essential for simulation and analysis of thermal management policies. This chapter first provides an overview of the related work in temperature and reliability modeling. Section 2.2 proposes a novel approach for accurate and fast temperature and reliability modeling at architecture level for multiprocessor systems. We discuss the experimental methodology in Section 2.3. In Sections 2.4 and 2.5, we show how to use the novel simulation framework for analyzing the effects of runtime management techniques on reliability, performance, and energy.

2.1 Related Work

Traditionally, SoCs and thermal packages have been designed considering the worst case temperature that can be reached during execution. As designing for the worst-case is getting prohibitively expensive with every process technology, detailed thermal modeling has become a requirement for both thermal and reliability management of systems. Temperature and reliability modeling enables evaluating and implementing both design-time and run-time temperature optimization methods. This way, we can reduce the area and performance costs of cooling solutions.

Thermal modeling is typically accomplished by constructing an equivalent RC network of the chip. Heat flow is analogous to the current passing through a thermal

resistance in the RC network. The transient behavior of temperature is modeled by means of the thermal capacitance. Compact thermal modeling tools, such as HotSpot, address the need for detailed thermal analysis (e.g., [59]). HotSpot models the vertical and horizontal thermal resistances and capacitances automatically for the given floorplan and package information. It takes into account the lateral thermal diffusion on chip as well. To speed-up the thermal modeling process, which is computationally costly, Atienza et al. introduce an FPGA-based fast thermal emulation framework [3]. This framework also constructs the RC network for the given chip, and the emulation platform can reduce the simulation time considerably in comparison to simulation while maintaining accuracy.

Few papers in the literature have taken reliability explicitly into account. Reliability management has been mostly addressed previously as a way of optimizing the policies or architecture at design-time (e.g., [62]). The Reliability-Aware Microprocessor (RAMP) provides a reliability model at the architecture level for temperature-induced intrinsic hard failures [62]. RAMP analyzes the effects of application behavior on reliability, and enables optimizing the architectural configuration and the voltage/frequency setting at design time to meet the reliability target. Previous work also shows that aggressive power management can adversely affect reliability due to fast thermal cycles, and optimization methods that consider reliability constraints provide energy savings while improving the MPSoC lifetime [52].

The SimPoint tool [57] addresses the problem of long simulation times, and it provides clustering analysis to identify a few representative points that can be simulated to predict the performance of the entire application. Biesbrouck et al. [5] use individual program phase information (a complete phase trace) to guide multithreaded simulation. This is accomplished by creating a *Co-Phase Matrix*, which represents the per-thread performance for each potential combination of the single-threaded phase behaviors that occur when multiple programs are run together.

Instead of summarizing the application behavior as in SimPoint, we need to capture the entire behavior to perform meaningful reliability analysis. The work presented in this chapter uses SimPoint's phase identification mechanism to capture a complete phase trace as part of the simulation process. This phase-based framework is able to simulate much longer periods of real-life execution in reasonable time frames.

2.2 Phase-Based Reliability Simulation

Simultaneous modeling of performance and reliability presents new methodological challenges that require tools and solutions radically different than traditional architectural investigation. This section describes the entire simulation infrastructure with a specific focus on the two most novel aspects of the framework: long time-frame performance estimation and the integrated reliability simulation.

To analyze runtime management techniques fairly, we need a fully integrated performance, power, and thermal model of the entire chip multiprocessor. This is because we are interested in evaluating management techniques that observe the temperature and power characteristics of the processor, and make management decisions accordingly. Also, we want to capture various types of effects that the architectural simulation provides; e.g., workload-dependent utilization of specific architectural structures and their impact on power and temperature, the time-varying behavior of individual applications, etc. However, thermal events that affect reliability, such as thermal cycles, happen over larger time scales than architecture-level simulation. Architecture-level simulators typically evaluate system behavior at instruction-level and run a total of several hundred million instructions. Interactive architecture-level simulation of the full benchmark is not practical, as just a single detailed architecture-level simulation corresponding to several minutes of real execution time requires weeks or months to complete. Therefore, we introduce new performance modeling mechanisms, integrated with our power, thermal, and reliability models, that allows accurate modeling of execution behavior over tens or hundreds of seconds.

Our simulation framework is shown in Figure 2.1. The performance modeling front-end combines a full-program phase profile with detailed architecture-level simulation of every distinct program phase at all possible frequency settings, including both performance and power characteristics. This characterization is gathered in a database that can be later queried while the full MPSoC simulation progresses. This way, we can model the effects of changing frequency, migrating jobs, etc., without further architecture-level simulation. After scheduling decisions are made and the resulting performance and power data are produced, we can model time-varying temperature effects across the entire chip. The temperature curves are then fed into the reliability models, producing the expected failure rates.

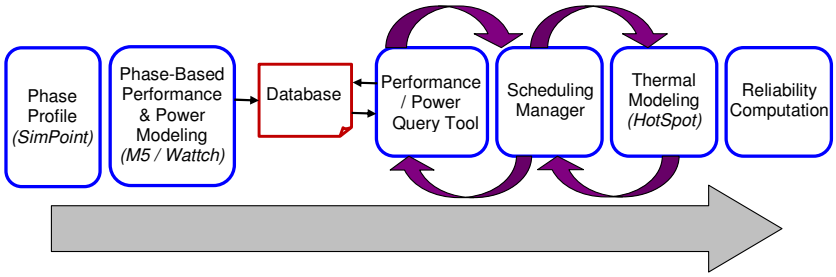

Figure 2.1: Design Flow

2.2.1 Long-Term Performance Modeling

Our framework relies on two simplifying assumptions that are critical to making this problem tractable. The first is that the time constants over which temperature varies do not require us to fully capture cycle-by-cycle variances in the temperature portion of the model. The instruction-level variations are captured in the performance model, but only summarized in the latter stages. This allows us to replace the cycle-by-cycle data with a stair-step graph, presenting performance and power behavior as constant at the average values over individual intervals. Thus, we can capture the program behavior with little loss of accuracy.

The second assumption is that the behavior of individual threads is separable. This is accurate because we model systems with private L2 caches, which is a likely architectural scenario in future systems [42]. At 16 cores and above, the interconnect cost of a shared cache would be extremely high. This assumption has been demonstrated to be accurate even on research that does not require this type of long simulation [41]. Even with the small core counts in current multicore systems, the AMD dual-core and quad-core Opteron, the IBM Power6, and the coming Intel Nehalem processor all have private L1 and L2 caches. For shared L2 caches, interactions between threads is higher and system-level accuracy of the simulation framework is reduced.

A phase of an application is a segment of execution during which a measured program metric is relatively stable. In our framework, we capture a complete phase profile of each application, beginning to end. Then, using the performance simulator integrated with the power modeling tool, and utilizing a finite number of simulation samples for each phase, we reconstruct the power and execution properties of the complete program. In fact, we do this for all voltage and frequency settings available, so that we can reconstruct the complete program even in the face of an arbitrary number

of voltage/frequency changes.

We use SimPoint [57] methodology to identify the various phases within the applications and to characterize complete program execution. A program's execution is divided into intervals of 100 million instructions. Once we assign each interval to a representative phase, we represent a program's execution by a Phase-ID trace [5]. Thus, at any instruction during a program's execution, we use this file to determine the current phase and to identify points of transition between phases. By running simulations at each phase point in the performance simulator and composing performance/power statistics with the Phase-ID trace, we create both a power and a performance trace.

We capture these program traces in a database which can be queried by the *Scheduling Manager* (see Figure 2.1) at distinct intervals. Given a program start point, an interval length in cycles, and a frequency setting, the query tool returns the average instructions per second (IPC) and power levels across the interval, and the point the program reaches in execution at the end of the interval. Thus, at runtime, the scheduling manager can make decisions about thread migration, thread stalling, or voltage/frequency changes, and query the database to model the precise effects.

2.2.2 Power Modeling

Power modeling requires coupling the execution traces obtained from the performance simulator with a tool that computes the power consumption for each functional unit. This coupling converts the performance parameters (e.g., cache accesses, branch predictions, etc.) into estimates for transistor switching, and then the power model utilizes these estimates for calculating the instantaneous power values. An example of such power modeling tools is Wattch [12], which is also used in this work.

Transistors consume power when they switch output values, but they also leak power even when they do not switch. The former is referred to as dynamic power, and historically has been the dominant factor; however, as technology shrinks, leakage power becomes increasingly important. We compute the leakage power of CPU cores based on structure areas, temperature, and supply voltage. To account for the temperature and voltage effects on leakage, we use the second-order polynomial model proposed by Su et al. [66]. This model computes the change in leakage power for the given differential temperature and voltage values, assuming a base leakage power density of $0.5W/mm^2$ at 383K [8]. We determine the coefficients in the polynomial model empirically to match

the normalized leakage values in [66].

2.2.3 Thread Management

We design a scheduling manager which enables the simulation of a large array of thread management policies. The mechanisms available for managing temperature include adjusting the frequency/voltage of a core (DVFS or DVS), putting an idle core into a low-power sleep mode (DPM), migrating computation off of a hot core, and policies that stop activity on a hot core (i.e., clock- or fetch-gating). In each policy, the scheduling manager makes a set of decisions after each scheduling interval, and it may incorporate performance and thermal information from the prior interval. After making those decisions for each thread and core, the scheduling manager then queries the performance database to obtain the power and performance behavior of each core over the next interval. Our simulation sampling intervals (i.e., 50 ms) are shorter than a scheduling interval, so there would be multiple exchanges with the performance database before another scheduling decision is made.

Since the scheduling manager keeps track of performance and power information, it also has the responsibility of modeling complex phenomena such as the delay from thread migrations. The model simulates the effects on power and performance for the following phenomena: thread migration, DVFS, starting a new application on a core, core sleep, and core wakeup. Our assumptions for several of the delays modeled are presented in Table 2.1. We model two aspects of the cost of thread migration among cores. We measure the software overhead in M5's full system mode as the time for Linux to migrate a thread from one core to another idle core and to start execution. This thread migration takes less than 3.0 μs. We also attribute architecture overhead to cold start effects in the branch predictor, caches, TLBs, etc. We measure cold start effects by starting each benchmark at many different points in the program, and then computing the average loss in performance. The average loss is 204 μs, but the loss varies wildly by benchmark—i.e., from 2 to 740 μs. Note that cold start effects dominate the migration penalty. To address the highly variable overhead, we model a distinct migration penalty for each benchmark.

Table 2.1: Delay and Power Model Assumptions

Parameter	**Model Value**
Sampling Interval	50ms
Thread Migration Delay	syscall delay + cold start effects
DVFS Delay	syscall delay + 20E-6s
Wakeup Delay	25ms
Application Startup Delay	syscall delay + cold start effects
Transition Power (to and from sleep states)	10W

Table 2.2: HotSpot Parameters

Parameter	**Value**
Die Thickness	$0.1mm$
Core Area	$14.44mm^2$
L2 area (total of 2 banks)	$10.56mm^2$
Convection Capacitance	140 J/K
Convection Resistance	0.1 K/W

2.2.4 Thermal Modeling

Automated thermal modeling requires power traces for each unit as input, in addition to the chip and package characteristics such as die thickness, heat sink convection properties, etc. Therefore, we feed the detailed power trace derived by the combination of the scheduling manager and the performance/power database into the thermal model. We modify HotSpot's [59] (block model) settings to model the thermal characteristics of an 16-core die. We use the steady state temperature of each unit as the initial temperature values. We summarize the HotSpot parameters in Table 2.2. We calculate the die characteristics based on the trends reported for 65nm process technology [29].

2.2.5 Reliability Modeling

Once we generate a full thermal trace, we use this trace as input to our reliability model. Our work targets temperature-induced reliability problems. The most commonly studied temperature-induced intrinsic hard failure mechanisms are electromigration, time dependent dielectric breakdown, and thermal cycling [34, 62].

Electromigration (EM) occurs in interconnects as a result of the momentum transfer from electrons to ions that construct the interconnect lattice and leads to hard failures such as opens and shorts in metal lines. The EM failure rate (λ_{EM}), based on

Black's model, is given in Equation 2.1. In the equation, E_a is the activation energy, k is the Boltzmann's constant, T is the temperature, J and J_{crit} are the current density and the threshold current density, respectively, and A_0 is a material dependent constant. We represent the first half of the equation with the term λ^0_{EM}, which can be considered as a constant (an average technology/circuit dependent value).

$$\lambda_{EM} = A_0(J - J_{crit})^{-n} e^{(-E_a/kT)} = \lambda^0_{EM} e^{(-E_a/kT)} \tag{2.1}$$

Time dependent dielectric breakdown (TDDB) is a wear-out mechanism of the gate dielectric, and failure occurs when a conductive path is formed in the dielectric. TDDB is caused by the electric field and temperature, and the failure rate is defined in Equation 2.2. Similar to the EM failure rate equation, we use λ^0_{TDDB} to represent the first half of the equation. Both EM and TDDB failure rates are exponentially dependent on temperature.

$$\lambda_{TDDB} = A_0 e^{\gamma E_{ox}} e^{(-E_a/kT)} = \lambda^0_{TDDB} e^{(-E_a/kT)} \tag{2.2}$$

Thermal cycling (TC) is caused by the large difference in thermal expansion coefficients of metallic and dielectric materials, and leads to cracks and other permanent failures. The thermal cycling effect is modeled by the Coffin-Mason equation [34]. Slow thermal cycles happen because of low frequency power changes such as power on/off cycles. Fast cycles occur due to events such as power management decisions. Although lower frequency cycles have generally received more attention, recent work shows that thermal cycles due to power or workload variations can also degrade reliability [49, 52]. The failure rate due to thermal cycling is formulated as in Equation 2.3.

$$\lambda_{TC} = C_0(\Delta T - \Delta T_o)^{-q} f \tag{2.3}$$

In this equation, ΔT is the temperature cycling range. The elastic portion of the thermal cycle is shown as ΔT_o. Elastic thermal stress refers to reversible deformation occurring during a cycle, and ΔT_o should be subtracted from the total strain range. Typically, $\Delta T_o << \Delta T$ [34], so the ΔT_o component can be dropped from the equation.

C_0 is a material dependent constant, q is the Coffin-Manson exponent, and f is the frequency of thermal cycles. Note that the Coffin-Manson equation [34] computes the number of cycles to failure. Therefore, the mean-time-to-failure (MTTF) in years is the number of cycles multiplied by the period of the cycles.

Computing the frequency of cycles is not straightforward in a simulation of an irregular, dynamic system. To resolve this problem, we observe the recent temperature history on each core to compute ΔT and f. We set the initial length of the history window to 5 seconds, and adjusted the length dynamically depending on how many cycles are observed. For example, if no temperature cycles are observed in the last interval, we increment the history window length to capture slower cycles. ΔT is the temperature differential observed in the last interval. We set a higher band of 80% and a lower band of 20% of the temperature range recorded in the last interval, and count the number of times the temperature exceeds the higher band or goes below the lower band, and use that to calculate the number of cycles in this period. In this way, we can account for the contribution of cycles with varying temperature differentials and varying periods.

For combining the failure rates, as in RAMP [62] we use the sum-of-failure-rates model. This model assumes that all the individual failure rates are independent. MTTF is $1/\lambda$ for constant failure rates; therefore, we average the failure rate observed throughout the simulation and compute the corresponding average MTTF. The average MTTF value reported for 65nm technology is 7 years [64].

For moderate temperatures at 65nm technology, Srinivasan, et al. [63] demonstrate that the contribution of electromigration, dielectric breakdown, and cycling to the overall failure rate are similar to each other. This allows us to calibrate the constants in each failure equation (λ^0_{EM}, λ^0_{TDDB}, and C_0) to give a system MTTF of 7 years at nominal temperature. We use the same constants all throughout the experiments, which means that the relative impact of different failure mechanisms might change depending on the conditions. For example, if the temperature is high, then the effect of EM or TDDB is higher than TC.

2.3 Methodology

This section describes other details of our simulation infrastructure that impact the results shown in the following sections. These, in general, are details that are rela-

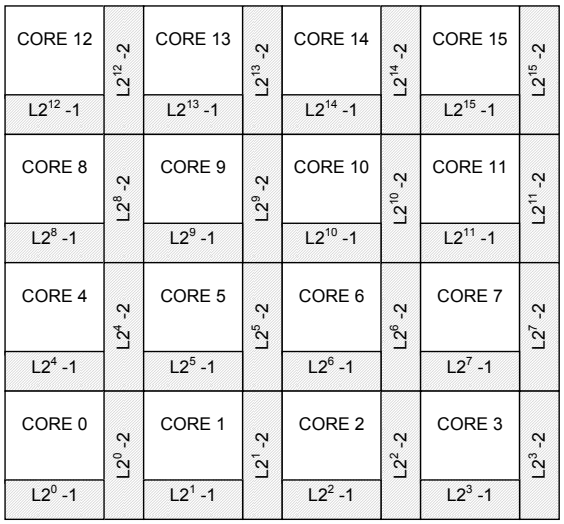

Figure 2.2: Floorplan of the 16-Core CPU

tively independent of our framework described in this chapter, such as the specific tools we use, processor core model, and the workload.

We use the M5 Simulator [6] for performance modeling. M5's out-of-order execution model is based on SimpleScalar 3.0 [13], and provides a detailed model of an Alpha 21264 processor. Anticipating continued scaling of core counts, the CPU we model is a 16-core multiprocessor manufactured at 65nm. The floorplan for this CPU is provided in Figure 2.2. Each core has out-of-order issue, a private data cache, instruction cache, L2 cache, and memory channels. Each core possesses three voltage and frequency settings for dynamic voltage/frequency scaling: 1.200V at 2.0GHz, 1.187V at 1.900GHz, and 1.06V at 1.7GHz which represent DVFS settings of 100% (original), 95% (step-1), and 85% (step-2), respectively. The architectural parameters of each core are depicted in Table 2.3.

We utilize Wattch [12] for the dynamic power modeling of cores in our framework. We integrate Wattch with M5 to provide dynamic and cycle accurate power measurements for each application. We develop a power model for 65nm by scaling the parameters within Wattch to match published power values for 65nm technology. The variation in dynamic power range we observe matches the power distribution on a similar core [31], on which the majority of applications (including SPEC) have less than 16% power dissipation difference from the other applications.

The leakage model we use is found to match closely with measurements [66], and

Table 2.3: Architectural Parameters

CPU Clock	2.0Ghz
ICache	64KB 2-way @1ns (2 cyc)
DCache	64KB 2-way @1ns (2 cyc)
L2Cache	2MB 8-way @10ns (20 cyc) (2 banks)
Memory Latency	100ns (200 cyc)
Branch Predictor	21264-style tournament predictor
Issue	out-of-order
ROB	128 entry
Issue Width	4
Functional Units	4 IntAlu, 2 IntMult, 2 FPALU, 2 FPMultDiv
Physical Regs	128 Int, 128 FP
IQ entries	64
Vdd	1.2V
DVFS Settings	100%, 95%, 85%

the leakage values produced in our work are in line with expected values (i.e., 30–40% of the total power consumption for 65nm).

To model power dissipation of L2 caches, we use CACTI [67] (an integrated memory performance, area, leakage, and dynamic power model) and obtain the typical power consumption of a memory block with the given size and properties, and then use these values throughout the simulation.

To create representative workloads, we classify all SPEC2K benchmarks in terms of their variability and memory boundedness. The distinction between CPU bound and memory bound applications is particularly important in this study because it impacts how performance scales as the frequency changes. We model both homogeneous and heterogeneous workloads in terms of the applications' CPU or memory boundedness. As our execution model does not extend to parallel programs, the homogeneous workloads stand in for both homogeneous server-type workloads and parallel applications with few stalls for communication. However, our homogeneous and heterogeneous multiprogrammed workloads best represent a server environment, where the average lifetime of the processor can significantly affect overall costs.

We use the ratio of memory-bus transactions to instructions as a metric to classify applications as memory or CPU-bound, as suggested by Wu et al. [72]. We classify applications along several other dimensions. By constructing our workloads from applications with different phase variability, power savings potential and CPU/memory boundedness, we seek to represent a wide range of real world workloads.

Table 2.4: Workload Characteristics

Wkload name	**Description**	**Cores Utilized**	**Benchmarks**
hom_16_cpu	Homogen. CPU Bound	16	sixtrack*16
hom_16_mem	Homogen. MEM Bound	16	mcf*16
het_16_cpu	Hetero. CPU Bound	16	mesa*3, bzip2_program*3, crafty*2, eon_rushmeier*3, vortex1*2, sixtrack*3
het_16_mem	Hetero. MEM Bound	16	mcf*4, art110*4, equake*3, gcc_166*3, swim*2
het_16_mix	Hetero. MIX	16	mcf*2, mesa, art110, sixtrack*2, equake, bzip2_program, eon_rushmeier*2, swim, applu, twolf, crafty, apsi, lucas
het_12_cpu	Hetero. CPU Bound	12	mesa*2, bzip2_program*3, crafty*2, eon_rushmeier*2, vortex1*1, sixtrack*2
het_14_cpu	Hetero. CPU Bound	14	mesa*2, bzip2_program*3, crafty*2, eon_rushmeier*2, vortex1*2, sixtrack*3
het_12_mix	Hetero. MIX	12	mcf*2, mesa, art110, sixtrack*2, swim eon_rushmeier*2, crafty, apsi, lucas
het_14_mix	Hetero. MIX	14	mcf*2, mesa, art110, sixtrack*2, eon_rushmeier*2, swim, twolf, crafty, apsi, lucas, equake

Table 2.4 describes each workload. We model workloads with 12–16 threads—our CMP architecture is constructed to not have thermal issues when lightly loaded, which is the expected behavior for the next few processor generations. We construct both homogeneous and heterogeneous workloads, and CPU-bound, memory-bound, and mixed workloads. The mixed workloads contain applications from both extremes, as well as some in the middle of our categorization. In the time frames we model, several of the applications complete execution. In those cases, we continually restart the application at the beginning to get consistent behavior across the experiment.

A common performance metric on multicore platforms is a raw count of IPC. However, this metric gives undeserved bias towards high-IPC threads as performance may be increased by running more CPU bound threads. To circumvent this difficulty, we use the Fair Speedup Metric (FS) [14, 61]. FS is computed by finding the harmonic mean of each thread's speed-up over a baseline policy of running the thread at the highest frequency and voltage. Although some applications repeat multiple times during

our simulations, we compute FS in such a way that the overall contribution of each application is the same.

2.4 Design and Implementation of Runtime Management Policies for Multicore Systems

The simulation infrastructure allows us to design and evaluate various job allocation and thermal management strategies. We divide these techniques into four categories: (1) power management policies that change the power consumption of a core by putting it into sleep or idle state, (2) policies that change what is running on a core via migration or scheduling, (3) policies that continue to execute the same thread but change speed (via DVFS), and (4) hybrid policies that combine DVFS and scheduling. The management policies evaluate the system characteristics at every scheduling period, and make a decision accordingly. In all cases, the scheduling tick is set to every 200ms. The threshold temperature for all the temperature-triggered policies is 85^oC. The **default** policy keeps the initial assignment of jobs to cores fixed, and no workload migration or voltage/frequency scaling occurs on the fly.

2.4.1 Power Management Policies

Dynamic Power Management (DPM) is one of the techniques we investigate to manage power. DPM puts cores in sleep state to save energy. We implement a fixed timeout policy [37], which is one of the commonly used DPM policies. For each core, the policy waits for a timeout period when the core is idle, and then turns off the core. This is to ensure that we do not turn off cores for very short idle times, where turning off the core would not amortize the cost of transitioning to and from the sleep state. The time period to amortize the cost of going to sleep is called the breakeven time (t_{be}). We assume a sleep state power value of 0.05W, and based on the active and idle power dissipation values we compute the t_{be} to be around 200ms. A simple and effective way to set the timeout period is $t_{timeout} = t_{be}$ [37]. DPM can be integrated with any of the other policies we discuss in this section for improving the energy savings.

Stop_Go [24] runs each core at the default (highest) frequency and voltage setting until a core reaches the thermal threshold. At this point, the core is stalled and the clock is gated to reduce power consumption. If the core's temperature goes below the

temperature threshold in the next sampling interval, execution continues. We assume that each core can be clock-gated individually.

2.4.2 Migration and Scheduling Policies

These techniques attempt to move computation off of hot cores either via migration or as a response to a thermal event (i.e., high temperature) as a matter of policy.

Migration sends jobs that have exceeded a thermal threshold to the coolest core that has not been assigned a new thread during the current scheduling period. If the coolest core selected is already running a job, we swap the jobs among the hot and cool cores. This technique can be thought of as an extension of core-hopping or activity migration techniques [26, 29] to the case of many cores and many threads.

Balance assigns jobs with the highest committed IPC during the last interval (i.e., between the last two scheduling ticks) to cores that have the lowest temperature. This scheduling idea represents a more proactive form of migration in which threads are dynamically assigned to locations before thermal thresholds necessitate action.

Balance_Location is similar to balance, but instead of assigning the threads with the highest committed IPC to the coolest cores, it assigns them to cores that are expected to be coolest based on location. The cores on the corner locations of the floorplan are expected to be the coolest; the remaining cores on the sides are expected to be the second coolest; and the cores in the center of the floorplan are hottest. This is because the temperature of a core is a result not only of activity on that core, but also on the activity of its neighbors: higher number of active neighbors results in hotter cores. Figure 2.3 shows the strategy we use to assign 16 jobs (j_1 to j_{16}) to cores, where the jobs have decreasing committed IPC ($IPC_1 > IPC_2 > ... > IPC_{16}$). Whereas the optimal allocation of threads to cores might diverge from the allocation shown in the figure depending on the IPC difference among threads, this allocation generally results in temperature characteristics close to the best allocation. With this scheme and 14 threads, for example, cores labeled j_{15} and j_{16} in this figure would always be idle.

2.4.3 Voltage/Frequency Scaling Policies

This set of techniques rely exclusively on dynamic voltage and frequency scaling to control thermal dynamics, but they differ in how and when DVFS is applied.

J_3	J_{10}	J_7	J_2
J_6	J_{14}	J_{15}	J_{11}
J_{12}	J_{16}	J_{13}	J_5
J_1	J_8	J_9	J_4

Figure 2.3: Thread Assignment Strategy for Balance Location

DVFS-Threshold (dvfs_t) reduces voltage and frequency (V/f) one step at a time when a core's temperature exceeds a threshold. After reducing the V/f to the step-1 (95%) setting, if the core is still above the threshold in the next scheduling interval, dvfs_t uses the step-2 (85%) setting. When a core's temperature is below the threshold, the V/f setting is increased, again one step at each scheduling interval.

DVFS-location (location_dvfs) uses a fixed V/f setting for each core, and there is no dynamic scaling at runtime. As the center cores tend to heat up more quickly, the four cores in the center of the floorplan have the 85% setting. The corner cores are typically the coolest cores, hence they use the 100% (original) V/f setting. The rest of the cores (i.e., the eight remaining cores on the sides) have the 95% setting.

DVFS-Performance (dvfs_perf) reduces the voltage and frequency dynamically on a core depending on the memory boundedness of the current application phase. Previous work shows that CPU-intensive tasks do not gain much in terms of energy savings from running at low frequencies; and conversely, it is beneficial to run memory-bound tasks at a lower frequency [23], as their performance is much more tolerant of frequency scaling. DVFS-Performance seeks to reduce the overall chip temperature with minimal performance cost by proactively scaling back those applications that are least impacted.

To determine the memory-bound phases, we use a cycles-per-instruction (CPI) based metric, μ, as defined by Dhiman et al. [23]. It compares the observed CPI with a potential CPI we might have gotten without memory events. If the μ is near one, the application is CPU-bound. If it is low, the application is memory-bound. Note that μ can also take negative values. Analysis on our own application set confirms that this metric tracks extremely well with performance degradation in the presence of DVFS.

If the μ observed in the last interval is less than -0.8, then we use the 85% setting,

and we find less than 6% performance loss during those phases. If $-0.8 < \mu < 0.5$, we apply the 95% setting, which induces less than 5% loss in performance. For $\mu > 0.5$, we do not perform any V/f scaling. When $\mu > 0.5$, if we used the 85% scaling for CPU-bound applications, the performance loss would be in the range of 12–15%.

DVFS-Performance_Threshold (dvfs_perf_t) behaves exactly like *dvfs_perf* unless a core reaches a thermal threshold. If the temperature exceeds the threshold on a core, then the policy activates *dvfs_t* to reduce the temperature on that core. After the core's temperature returns within threshold, we switch back to *dvfs_perf*. This technique is very successful if by proactively slowing a thread that is tolerant of frequency changes, it can enable a nearby thread that is not so tolerant of frequency change to avoid a DVFS slowdown.

2.4.4 Hybrid Techniques

In investigating the interaction of scheduling and DVFS policies, we employ Balance_Location to represent the scheduling policies. It has useful properties in terms of both reliability and performance. It does only enough migration to find the best location for each thread, then only migrates when application characteristics change.

Balance_Location & DVFS-Threshold works by initially using *Balance_Location* to assign potentially hotter threads to cooler locations on the die. If this technique fails to keep a given core under the specified threshold, the core employs *dvfs_t* until it is under the thermal threshold.

Balance_Location & DVFS_Performance uses the *Balance_Location* policy to allocate jobs to cores, and runs *dvfs_perf_t* at the same time to decide on the V/f settings of the cores.

Balance_Location & DVFS-Location assigns the location V/f settings as in the *location_dvfs* policy to cores, and performs *Balance_Location* for allocating the threads. This tends to have the effect of assigning the most memory-bound threads in the center zone, which runs at the 85% setting.

Balance_Location uses IPC in assigning threads to locations. When combined with DVFS, we must account for the V/f and its effect on the measured IPC. Thus, if a core is running at a lower V/f setting, we scale the measured IPC based on the average performance hit observed at that V/f level.

2.5 Experimental Results

In this section we demonstrate that the framework proposed in this chapter allows us to evaluate a large set of previously proposed and new scheduling algorithms, in terms of performance, power, temperature, and processor lifetime (reliability). Such a comprehensive evaluation would not have been possible with existing tools due to the long simulation times of traditional architecture-level simulators. We show that having a fully integrated framework, including a reliability model that accounts for all the major causes of temperature-induced hard failures, sheds some new light on multicore scheduling. The section starts with an evaluation of the accuracy of the simulation framework, and continues with evaluating the policies discussed in Section 2.4.

2.5.1 Accuracy

The described methodology allows us to do full-program simulation with simple look-ups of sampled simulation data. This sacrifices some accuracy. However, the rate at which temperature changes is typically slow when compared to even complete phases, so this technique actually sacrifices little accuracy. We validate our methodology by comparing the results with direct M5/ Wattch power output. For each phase simulation point of each SPEC benchmark, we run M5 and Wattch for 500ms of simulated execution and gather power statistics every $500\mu s$. We compare the power statistics of M5/Wattch and our framework, and we see that our phase-based approach has 1.8% average error overall. Table 2.5 shows the detailed results for the benchmarks. *bzip2* with input set *program* has the largest average error of 3.0%.

The low error margin in our power computation methodology translates to even lower error in temperature computation because of the thermal time constants. To verify the accuracy of our methodology in terms of the temperature response, we experiment with bzip2, as it has the highest power error margin. Figure 2.4 shows one particular (worst case) data point—the temperature trace for a core running bzip2 and then going to sleep state, on a system running 12 bzip2 threads. The "M5/Wattch" thermal trace corresponds to the detailed power trace sampled at $500\mu s$, and the "phase" trace is the thermal output of running the same workload and using our power computation methodology. We observe that the trace generated with our methodology closely matches the trace sampled at a higher granularity. As bzip2 is one of the most power-variant applications, the rest of the benchmarks demonstrate even less difference. Because thermal

Table 2.5: Power Estimation Error of Our Framework Compared to M5/Wattch

Benchmark	Average Error	Benchmark	Average Error
parser	0.023	facerec	0.022
applu	0.021	gcc_166	0.020
art110	0.016	fma3d	0.024
swim	0.018	mcf	0.011
galgel	0.015	gap	0.020
twolf	0.009	vpr_route	0.017
mesa	0.027	ammp	0.015
lucas	0.009	bzip2_program	0.030
vortex1	0.018	equake	0.029
sixtrack	0.011	eon_rushmeier	0.018
apsi	0.014	crafty	0.018
Overall: 0.018			

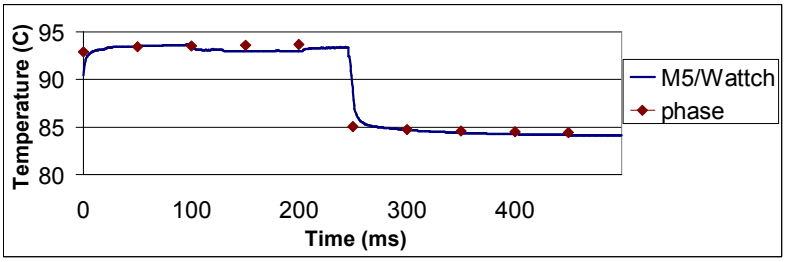

Figure 2.4: Comparison of Temperature Responses for *bzip2* Using Two Simulation Methodologies

cycling effects are insignificant unless the temperature variations are more than a few degrees, these results are more than accurate enough to capture both temperature-induced and cycle-induced effects.

2.5.2 Evaluation of Runtime Policies

Section 2.4 has identified a wide assortment of thread management, power management, and DVFS policies. In evaluating these policies in various execution scenarios, this section attempts to sort out the key issues facing the designer of a multicore thread management policy, such as: (1) how to properly combine scheduling/migration policies, DVFS policies, and DPM policies; (2) how to address peak temperature effects without exacerbating thermal cycling; (3) whether to use reactive or proactive DVFS policies; and (4) how to address thermal asymmetries in the chip multiprocessor.

We group the experiments in the following categories. We first look at full core

utilization scenarios with a varying number of memory-bound and CPU-bound threads, using the five 16-thread workloads from Table 2.4. For these experiments, threads are initially placed on the cores randomly (i.e., with neither a clearly good or bad initial allocation). Partial utilization results examines systems that are less than fully utilized, with 12 or 14 jobs (i.e., 2 or 4 idle cores). We then take a deeper look at the consequences of the initial allocation of idle cores. Finally, we investigate how reliability, performance, and energy vary when the system has DPM capabilities, and which schedulers best complement DPM to achieve the desired trade-offs for reliability, energy, and performance.

We present the energy and performance of the policies in addition to reliability. All results are normalized with respect to the *default* case of no thermal management (i.e., all threads running full speed on the initially assigned cores), allowing us to evaluate each policy on the same scale. In a real-life system, the *default* policy is typically coupled with a back-up thermal management policy to ensure the chip does not exceed critical temperatures. However, in this work, we do not integrate any thermal management policy with the default policy to use the default case as a reference point especially for performance.

Srinivasan et al. [64] report the average MTTF of the SPEC suite simulated for 65nm at 1.0V of supply voltage as 7 years, and our model is calibrated to the same value. However, if the reliability model was re-calibrated to assume a shorter or longer MTTF, the policies are expected to display similar trends, while the absolute numbers would change depending on process technology, baseline MTTF, and the system being modeled. Therefore, we show results based on the % change in MTTF values, rather than absolute numbers.

Full Utilization:

We first analyze the workload allocation policies' ability to improve thermal characteristics for workloads fully utilizing the system. The results in Figure 2.5, which are the average values for all the 16-thread workloads, indicate that *Migration*, *Balance*, and *Balance_Location* have little impact on reliability in this scenario—this is because cores are fully utilized and most of the workloads are highly homogeneous. Therefore, re-allocating workload does not change the temperature and reliability considerably.

In the one heterogeneous workload, the effect of workload allocation is still small. In that case, the *Balance* and *Balance_Location* policies each improve reliability by 4.4%

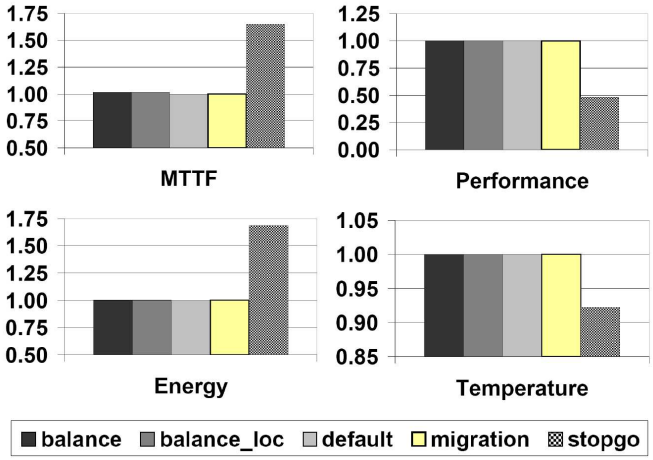

Figure 2.5: Comparison of Workload Allocation Techniques

with minimal impact (less than 1%) on performance, energy, and average temperature. Thus, in the absence of idle cores, these policies are less effective than the ones with voltage and frequency scaling.

The *Stop_Go* policy is notably different than the policies discussed above, as it has the ability to cool a core even in the absence of idle cores. *Stop_Go* improve the MTTF by 65%, but with a hefty 52% decrease in performance and a 69% increase in energy consumption. Average temperature of the processor is reduced by 8%. The *Stop_Go* policy is prone to creating large temperature variations due to switching among active and idle states. However, the frequency of stalling/resuming execution is high enough that the temperature variations are of a relatively low magnitude, and the reliability of the core is dominated by the thermal hot spots only (i.e., no significant increase in cycling-based failures).

Figure 2.6 shows the effect of the DVFS policies on reliability when the cores are fully utilized. DVFS has a much more significant impact than the workload allocation policies, due to its ability to reduce temperature even in the face of full utilization. In particular, we find several key insights in these results. First, it is important to always keep an eye on peak temperature. The *dvfs_perf* policy, by selectively choosing which threads to scale, sacrifices very little in performance, but does fall behind in MTTF in comparison to other DVFS policies. This is because it does not utilize thermal feedback from the system. The *dvfs_perf_t* policy reacts upon reaching a thermal threshold in addition to performance-aware DVFS, and as a result loses some performance, but it has

one of the lowest failure rates.

Second, we see significant benefits of proactive techniques over traditional reactive techniques. It is interesting to note that the other DVFS policies beat *dvfs_t* along all axes. This result is particularly surprising on the performance front, because: (1) *dvfs_t* only scales when it *has* to upon reaching a threshold, and (2) the other DVFS policies also behave the same as *dvfs_t* upon reaching the threshold temperature. The reason that other DVFS policies perform better is that proactively scaling a thread (whose performance is tolerant to scaling) reduces the temperature in that area, and often prevents other neighboring threads from reaching the threshold.

Third, we see that it is critical that our thread management policy understands the inherent thermal asymmetry of the multicore system. The policy that provides the best balance among all three metrics is *location_dvfs*, with a failure rate that is half of the baseline and a minimal performance loss (3.8% of default). To further investigate the policies' abilities to account for the asymmetries, we compare *location_dvfs* with homogeneous proactive scaling: all cores at *85% DVFS* and all cores at 95% DVFS. Among these DVFS techniques, *location_dvfs* still demonstrate the best trade-off point. The 95%-DVFS result improves performance over *location_dvfs* by less than 1%, but gives up 25% in processor lifetime. The 85%-DVFS increases reliability significantly, but more than doubles the performance cost compared to *location_dvfs*.

We examine the hybrid techniques in Figure 2.7. When we compare the hybrid policies against the DVFS based policies, we see that DVFS-based policies are improved little by combining them with job allocation policies. Again, this is due to the limited gains from reorganizing running threads on a fully utilized system.

The result on a fully utilized system shows DVFS policies or hybrid strategies have more substantial benefits in improving MTTF in comparison to workload allocation policies. The *location_dvfs* policy achieves the best trade-off among performance, reliability, and energy due to its ability to consider the thermal asymmetries in the 16-core system. Dynamic management techniques that take into account the thermal asymmetries in multicore chips are easily adapted to other sources of asymmetry, such as process variations, as long as we can quantify the effects of such variations on the thermal and power properties of each core.

Partial Utilization:

It is expected that most multicore systems will be utilized less than 100% most

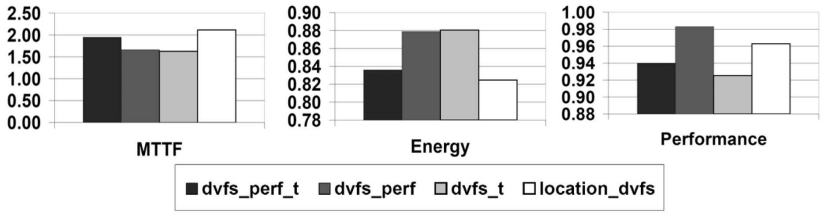

Figure 2.6: Comparison of DVFS-Based Techniques

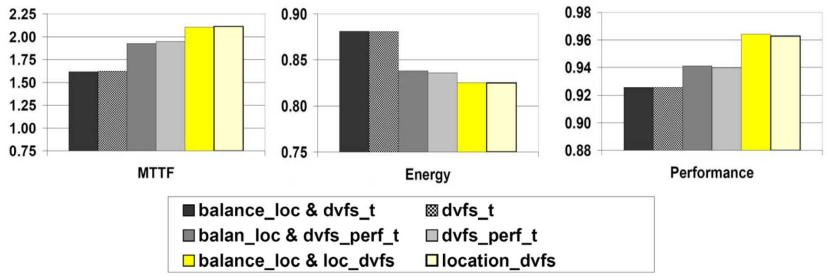

Figure 2.7: Comparison of Hybrid Techniques

of the time. This is true for the CMPs in the server domain as well as in personal computing. To evaluate the impact of scheduling mechanisms on reliability when some cores are idle, we use the 12 and 14 thread workloads described in Table 2.4: a CPU-bound and a mixed CPU-bound/memory-bound workload for each of the 12 and 14 thread cases. The results represent the average of the CPU-bound and mixed cases for the 12 and 14 thread experiments. At the beginning of each simulation, we decide which cores to leave idle by choosing the allocation with the lowest peak temperature. Once we determined the active cores, we perform the initial placement of threads on these cores randomly.

We first focus on the case with 14 active threads in Figure 2.8. Although this utilization is close to the full utilization examples earlier, the impact on the reliability of the various policies changes significantly. Policies with frequent workload re-allocation (i.e., *Balance*, *Migration*) result in lower reliability with respect to the other policies. The *Balance* policy assigns jobs to cores based on temperature rather than location and often mistakes a core that is cool now for a core that will stay cool in the future. Migration policies that focus heavily on current temperatures are prone to this type of inefficiency in management. The *Migration* result has the same issue. Policies that migrate more than necessary have two distinct reliability disadvantages over the other

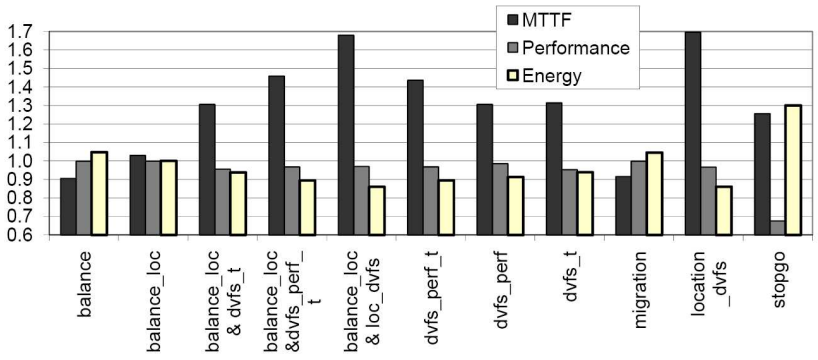

Figure 2.8: Effect of System Utilization (2 Idle Cores)

techniques. First, migrating too often will tend to thwart the DPM manager, which does not put a core to sleep until it has been idle for awhile. This increases the time cores spend running at hotter temperatures. Second, migration causes thermal cycling. This is the dominant cause of the low MTTF results, as the power variations between idle and active states start creating cycles of a noticeable magnitude.

The *Stop_Go* policy achieves 1.25 times improvement in MTTF. However, this comes at the cost of a drastic performance and energy cost. Although *Stop_Go* can be utilized effectively as a back-up policy for thermal emergencies to guarantee that temperature does not exceed a given peak value, it is inefficient if used frequently. Interactions between *Stop_Go* and DPM are discussed later in this section.

Among the DVFS policies, *dvfs_perf* achieves the best performance of less than 2% degradation, while *location_dvfs* results in the longest system life time with a 69% improvement. The hybrid policy *Balance_Location&location_dvfs* seems to provide the best trade-off point among the policies as it achieves almost the same MTTF as *location_dvfs* with better performance and lower energy consumption. The reason the hybrid scheduling policies still provide only small gains over DVFS policies alone is that we start the experiments with an optimal placement of idle cores.

We expect that as technology scaling continues, the bandwidth for performing voltage scaling will decrease due to the leakage power and transistor threshold voltage limitations. Therefore, future systems are expected to count on other mechanisms for managing power and temperature, such as workload scheduling.

Figure 2.10 shows the 12-core utilization results. Because chip temperatures are lower overall, the magnitude of potential reliability gains is reduced. In fact, policies

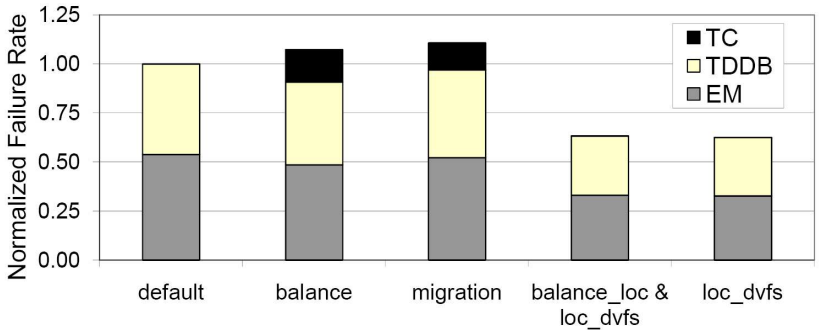

Figure 2.9: Contributions of Failure Mechanisms

that only react to thermal thresholds do not change the workload allocation or the V/f setting in this scenario (e.g., *Migration*, *dvfs_t*, etc.) and they give the same results as the default policy, as the core temperatures do not exceed the threshold. Policies that proactively look for opportunities can still improve processor lifetime significantly, and even *Balance_Location* provides small gains. Policies that proactively migrate based on current temperature (*Balance*) make mistakes and create thermal cycling.

In Table 2.6 we show the number of migrations and number of V/f setting changes per second for the policies to provide a more complete understanding of the runtime behavior. The policies that are not listed do not utilize migrations or DVFS. The columns marked as *ALL, corner, center*, and *side* refer to the average number across all cores, across only the corner cores, center cores and side cores, respectively. The results are with DPM, and for the CPU-bound heterogeneous workload with 14 threads (i.e., 2 idle cores). *Migration* has a significantly higher number of thread movements in comparison to other policies: almost 3 times more than *Balance_Location*. The low migration count of *Balance_Location* is a result of its ability to match the performance characteristics of applications with the thermal behavior of cores. Compared to *Balance_Location* only, combining *Balance_Location* with DVFS increases the frequency of migrations, as the temperature profile of the cores vary more when their V/f settings are dynamically adjusted. Among the DVFS policies, *dvfs_perf* has the lowest number of changes as it only alters the V/f setting of applications tolerant to operating at a slower speed. Also, *dvfs_perf_t* reduces the frequency of changes in comparison to *dvfs_t* as it proactively adjusts the V/f setting and triggers the thermal threshold fewer times.

Figure 2.9 presents a breakdown of the contribution of different failure types to reliability to better explain the tension between the different failure mechanisms.

Table 2.6: Number of Migrations and V/f Changes (per Second)

	Migrations			
	ALL	corner	center	side
balance	4.76	4.75	5.00	4.65
migration	7.66	8.36	5.00	8.66
balance_loc	2.73	1.25	1.60	4.03
balance_loc& dvfs_t	3.65	3.85	2.10	4.33
balance_loc& dvfs_perf_t	3.64	3.86	2.10	4.30
balance_loc& loc_dvfs	3.54	3.70	2.20	4.12
	V/f Setting Changes			
dvfs_perf_t	2.98	2.60	0.80	4.20
dvfs_perf	0.83	1.40	0.00	1.00
dvfs_t	3.40	3.60	0.40	4.80
balance_loc& dvfs_t	3.64	4.40	1.10	4.53
balance_loc& dvfs_perf_t	3.58	4.00	2.00	4.20

This figure demonstrates the normalized average failure rate for our two best and two worst policies (i.e., best/worst in terms of their average MTTF results), all integrated with DPM. The workload for this experiment is the heterogeneous CPU-bound workload with 12 threads. Recall that the failure rate is inversely proportional to MTTF. This figure shows that *Balance* and *Migration* reduce the probability of failures due to electro-migration (EM) and dielectric breakdown (TDDB). If we ignored the effect of thermal cycles, we would conclude that reliability has increased. However, because of the number of thread migrations, they create large thermal cycles (TC). The *location_dvfs* and the hybrid *Balance_Location&location_dvfs* policies, on the other hand, reduce the failure rates caused by thermal hot spots without introducing a significant amount of thermal cycling failures. Note that in the default case, as there is no workload re-allocation, temperature is stable and no cycles are observed.

On a single-threaded system such as the 16-core architecture we are using, when there are any number of idle cores in the MPSoC, workload allocation is able to help signifcantly with improving reliability at a lower performance cost in comparison to DVFS policies. In fact, combining *location_dvfs* and *Balance_Location* into a hybrid policy gives the best trade-off between performance and MTTF.

Effect of Initial Workload Allocation:

We also examine each policy's ability to adapt to different initial workload mappings on the processor topology. For example, what happens when the initial mapping of threads to cores is highly suboptimal? This could happen with a topology-ignorant

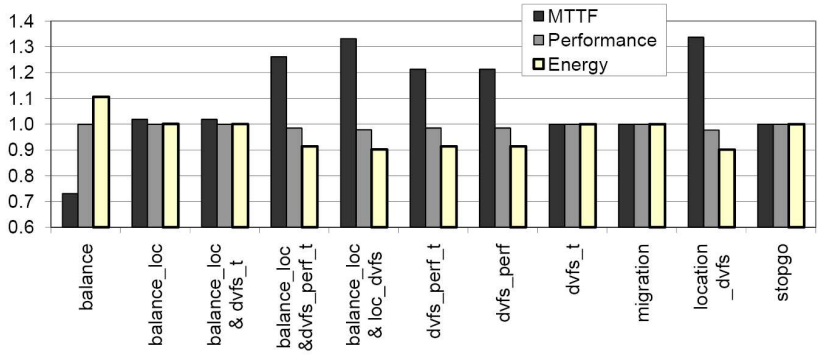

Figure 2.10: Effect of System Utilization (4 Idle Cores)

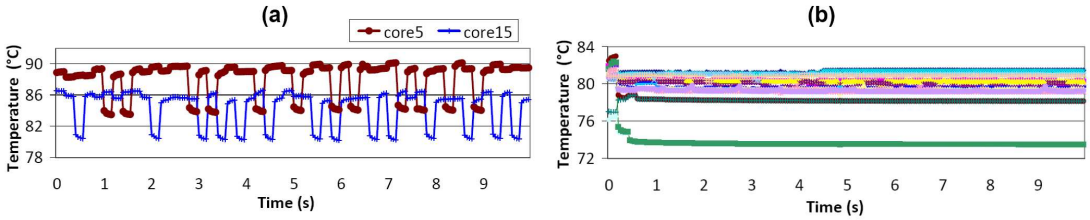

Figure 2.11: (a) Cycles Caused by the *Migration* Policy; (b) Stable Thermal Profile of *Balance_Location & location_dvfs*

scheduler (a likely scenario early on), or just because of jobs entering or leaving the system. We examine several ways of performing the initial job allocation: best possible, worst possible, and an in-order placement of jobs on cores.

The *best* case for 12 active threads, i.e., the case with the lowest peak temperature, is leaving the center cores (5, 6, 9, 10) idle, and for 14 active threads when cores 6 and 9 are idle. The *worst* case occurs when the corner cores are idle. Specifically, the worst case for a system with 12 active cores is leaving the cores 0, 3, 12, and 15 idle. Similarly, when 14 cores are active, leaving two of the corner cores on the opposite sides idle, such as cores 0 and 15, represent the worst assignment. The *in-order* initial assignment allocates all available threads on the cores starting from core 0 ascending. This method initially leaves cores 12–15 idle when 12 threads are active, and cores 14–15 idle with 14 threads are active. The *in-order* method attempts to model a naive scheduler that assigns jobs to cores using a *first-available* strategy.

We observe notable differences in reliability among the policies in the initial allocation experiments. For example, the policy *dvfs_perf_t* experiences a 15% reduction in MTTF in comparison to the best allocation when either the worst or in-order idle core

locations are used. This decrease in reliability is comparable to the default policy's 20% reduction in MTTF when using the in-order and worst case initial assignments. On the other hand, when *dvfs_perf_t* is combined with *Balance_Location*, we are able to achieve a level of reliability to match that of the optimal initial placement. This indicates that one of the major roles of the allocation policy is reassigning thread topologies to assist other policies that set core voltage and frequency. Thus, in a real CMP system, it is critical to combine a *conservative* migration technique (i.e., one which avoids unnecessary migrations and does not create cycling) with DVFS techniques. In the absence of an intelligent migration and scheduling policy, it is difficult to avoid detrimental configurations over time.

Interactions with DPM:

The last set of results are for dynamic power management (DPM), which takes advantage of prolonged core idleness to put the core into a sleep mode. The power consumption of the core is greatly diminished in sleep state. Each of the policies presented in Section 2.4 can be integrated with dynamic power management, but some are able to use DPM opportunities better. Taking a closer look at two extremes, we first examine two policies, *Migration* and *Balance_Location&location_dvfs* for the het_12_mix workload with 12 CPU and memory bound threads. Comparing the thermal traces for *Migration* and *Balance_Location&location_dvfs* (Figure 2.11), the *Migration* policy suffers significant thermal cycle variations. For *Migration*, we demonstrate the thermal cycles observed on two cores due to frequent re-allocation of workloads. For the *Balance_Location&location_dvfs* policy, we show all the cores' thermal traces, and observe that each core's temperature is stable and lower than the threshold.

This stability along with a lower peak temperature results in significantly higher reliability. *Balance_Location&location_dvfs* turns out to be the best policy when paired with DPM, and it provides an increase in MTTF of 36% over *Migration*, while the performance difference is only 1.5%. So we see that scheduling policies which effectively manage thread locations and DPM policies can reduce processor temperatures and improve reliability. DPM can also lead to greater thermal cycling which can counteract some of the MTTF gains that result from the lower power levels of sleeping cores. The adverse effect of DPM on reliability due to thermal cycles is also emphasized in previous work [52]. Thus, when we include the effects of thermal cycling failures, we observe that the traditional assumptions for developing management strategies are incomplete. The

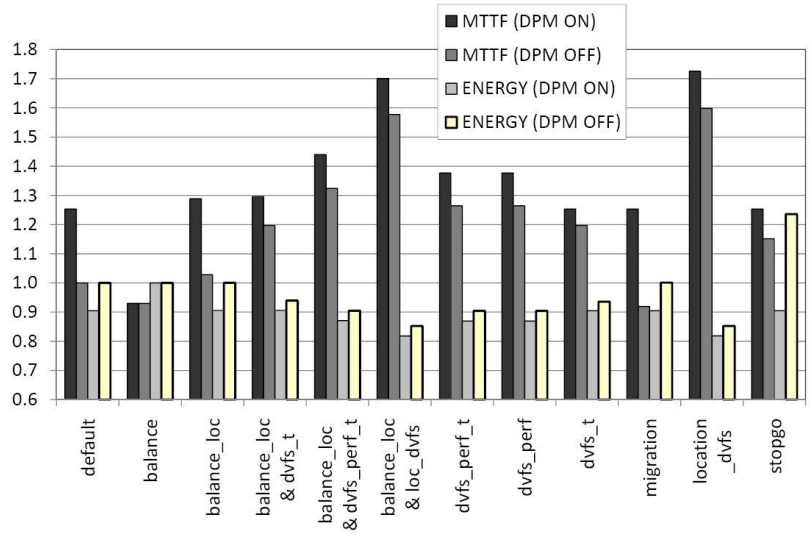

Figure 2.12: MTTF and Energy Effects of DPM

reliability effects of DPM should be considered while designing thermal management policies.

Despite DPM's possible impact on reliability, we do see (Figure 2.12) that even in the face of this cycling phenomena, DPM is an overall win for all policies with the exception of *Balance*. In this figure, we show the average results over heterogeneous CPU-bound workloads. The *Balance* policy receives no benefit from using DPM, because its proactive mechanism that keeps moving hot threads to colder cores. The result is that no core is idle long enough to trigger the sleep mode. On the other end of the spectrum, *Migration* and *Balance_Location* show gains of 27% and 20% in MTTF on average. The reliability improvement in DVFS-based techniques are less prominent and range between 4%–8% MTTF increase.

The policy *Stop_Go* receives a large benefit in energy from using DPM mechanisms, reducing power consumption by 27% in comparison to the no-DPM case. The reason for this benefit is the frequent stall cycles that are utilized for saving energy. If confronted with a design choice that requires the simplicity of *Stop_Go*, DPM could help regain much of the energy lost from the constant start and stop of individual cores.

2.6 Summary

This chapter proposes a novel multiprocessor simulation framework that is able to simulate thermal dynamics over far longer time periods than typical architectural simulators at high accuracy. For example, an architecture-level simulator may take days or weeks to simulate a few minutes of real execution time, whereas our framework is able to provide accurate reliability and performance results within hours. Using this framework, we have analyzed how job scheduling and power management policies affect system lifetime, energy, and performance.

The results in this chapter provide several key insights that help us understand the dynamics of single-chip multiprocessors and develop efficient management policies:

- Understanding thermal asymmetries, which are either due to the layout of the processor or due to process variation, is an important component of effective thermal management. Not considering this thermal disparity causes much unnecessary thread movement because we cannot discern between a hot thread and a hot core. Understanding the thermal variance allows us to employ an asymmetric thermal policy that accounts for and even exploits that asymmetry.

- Our most effective policy that employs voltage/frequency scaling (i.e., *location_dvfs*), as well as our best one that does not (i.e., *Balance_Location*), both account for the location asymmetry and reduce the number of thread movements. We have presented new scheduling policies in Section 2.4, such as *Balance_Location & location_dvfs*, that decrease the failure rate by a factor of two (over naive management), with a performance cost of less than 4%.

- It is critical to consider thermal cycling effects in addition to peak temperature effects. In Section 2.5, we see two policies that erroneously appear to increase lifetime when the effect of thermal cycling is ignored. Thermal cycling is not a significant effect in a fully utilized system, as the variance in power between running threads are not sufficiently high to cause harmful effects. However, when cores are idle, it is important that we manage the idle cores in a way that does not exacerbate thermal cycling. Considering typical server workloads rarely fully utilize a system, this observation is critical.

- Conservative policies that minimize migration not only reduce thermal cycling, but also maximize our ability to exploit sleep states via DPM. In addition, reducing migrations

helps reducing the performance overhead due to cold start effects.

- Proactive techniques that apply DVFS to frequency-tolerant applications raise the performance of the entire system. This is somewhat counter-intuitive, as the frequency-tolerant applications are also the coolest applications. However, by lowering overall temperatures chip-wide, proactive techniques allow the hot applications to run longer without triggering thermal events.

The text of Chapter 2 is in part a reprint of the material from the paper, *Ayse K. Coskun, Richard Strong, Dean Tullsen and Tajana Rosing, "Evaluating the Impact of Job Scheduling and Power Management on Processor Lifetime for Chip Multiprocessors"*, in Proceedings of SIGMETRICS/Performance 2009–Joint International Conference on Measurement and Modeling of Computer Systems, 2009. The dissertation author was the primary researcher and author, and the co-authors involved in the publication [22] directed, supervised, and assisted in the research which forms the basis for that material.

Chapter 3

Static Temperature-Aware Job Scheduling

Job scheduling has a significant impact on temperature and system reliability, as discussed in Chapter 2. This chapter proposes a static (design-time) technique for temperature-aware job allocation. This optimization technique is useful for two purposes: (1) we can use it as a baseline for evaluating dynamic scheduling policies; (2) for systems with *a priori* known workloads, such as some embedded systems, we can optimize the job schedule to minimize the temperature-induced challenges.

The goal of our static task scheduling approach is minimizing the frequency of thermal hot spots and large temperature gradients in order to increase system reliability and ease the temperature related design challenges. In addition to our technique, we formulate static scheduling for minimizing energy, balancing energy and minimizing the thermal hot spots (i.e., without considering temperature gradients). This way, for the first time, we provide a comparison of various optimal scheduling techniques in terms of their efficiency in handling thermal problems.

3.1 Static Optimization for Minimizing Hot Spots and Gradients

Our goal in static optimization is finding a task schedule for the MPSoC where the deadline and dependence constraints of tasks are met, and the best possible temperature profile is achieved throughout the execution. Achieving the best temperature profile

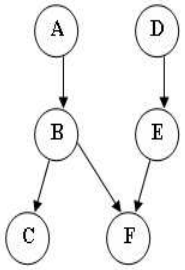

Figure 3.1: Example Task Graph

corresponds to minimizing and balancing the temperature across the MPSoC as discussed in Chapter 1. Addressing both the hot spots and gradients is our solution's distinguishing feature from the other energy and thermal based static scheduling strategies. Our method utilizes integer linear programming (ILP), which is commonly used for solving scheduling problems (e.g., [53]). Our ILP formulation guarantees to minimize the hot spots and gradients for the given assumptions of task execution times, thermal estimates of tasks, deadlines, and precedence constraints.

In our system model, we assume the MPSoC contains m processor units, $PU = PU_p;\ p = 1, ..., m$, and we model the applications using task graphs. In the graph $G = (T, E)$, each vertex represents a task ($T_i \in T$), which is a function or collection of functions to be performed. Each edge in the graph ($E_{ij} \in E$) shows that task T_j is dependent on T_i in order to perform computation. A simple example task graph is shown in 3.1. We assume the deadlines (D_i) and worst-case execution times ($WCET_i$) of tasks are known *a priori*.

We assume each PU has v discrete voltage settings $V = V_k;\ k = 1, ..., v$ (in decreasing order). Each voltage setting (V_k) can be associated with a computation speed f_k in terms of cycles/second. Thus, the energy consumption for executing T_i at speed f_k can be expressed as $e_{ik} = (g_{active}(f_k) \cdot t_i)$, where g_{active} is the function for power consumption and t_i is the execution time of T_i. The power consumed during task execution is a monotonically increasing and convex function of the computation speed [74]. Assuming the tasks execute up to their WCET and $WCET_i$ is given as the execution time in the default (highest) processor frequency, the WCET for T_i running at processor speed f_k is computed as: $t_i = WCET_i \cdot (f_1/f_k)$.

We provide the objective functions of all the ILPs we solve in this work in Table 3.1. Minimizing and balancing the hot spots (`Min-Th`) reduces the thermal hot spots,

but there is no consideration for spatial gradients. For energy balancing (`Bal-En`), the ILP minimizes the maximum energy consumption for each core, which balances the energy profile on the system. The ILP for minimizing energy (`Min-En`) minimizes the cumulative sum of all the active and idle state energy. The objective function of our ILP (`Min-Th&Sp`) has two parts: 1) Minimizing and balancing the thermal hot spots (H in Table 3.1); 2) Minimizing the spatial gradients (G). The first part of the objective function minimizes the maximum time spent above the threshold for each core, which balances the thermal hot spots across the chip. Consequently, it reduces the magnitude of temporal variations in temperature. However, it does not consider the spatial gradients on the die.

Spatial gradients increase when several jobs are clustered in neighboring units, while the rest of the units are idle. Contrarily, when the workload is spatially spread out on the die, because of the effect of heat transfer from hot to cool cores, more even temperature distributions are achieved. For example, a checkerboard arrangement of the workload where each active core has an idle neighbor reduces the spatial gradients in comparison to having all active cores on one half of the MPSoC and idles ones on the other half. Therefore, reducing the high spatial differentials requires avoiding scheduling jobs in neighboring cores at the same time. We call this situation *overlap*, where two jobs are scheduled in next-door neighbors at the same time. Thus, in the second part of the objective function (G in Table 3.1), we minimize the total overlap. We only take into account side-by-side neighbors in our overlap computations, as it is shown that heat sharing among neighbor units with adjoining sides have a significant effect on temperature [59]. Minimizing the sum of H and G addresses both balancing thermal hot spots and minimizing the spatial gradients.

Table 3.2 provides the complete formulation of the ILP for `Min-Th&Sp`, and the variables used in the formulation are defined in Table 3.3. In the first part of objective function, (H), we minimize the total time spent above a given threshold in order to eliminate the hot spots and balance temperature. We use "time spent over a temperature threshold" (represented with q_{ik}) as a metric for profiling the thermal behavior of tasks, and also as a metric for evaluating our ILP. This metric has also been used in previous work as well [43]. Minimizing the average or peak temperature does not adequately capture the temperature profiles. For example, a system that experiences 90^oC temperature for an hour is expected to have worse reliability than a system that has 90^oC

Table 3.1: Summary of All the ILP Objective Functions

Label & ILP Objective	**Objective Equation**
Min-Th&Sp : Minimizing thermal hot spots and gradients	Minimize $H + G$; $H = max\{Q_p; p = 1...m$, for a system of m cores$\}$ where: $Q_p = \sum_{T_i \in T} \{x_{ip} \sum_{f_k} (q_{ik} y_{ik})\}$ $G = \sum_{p,r \in PU, p \neq r} \{n_{pr} \{ \sum_{i,j \in T, i \neq j} x_{ip} x_{jr} [p_{ij} d_{ij} (\tau_i - s_j) + p_{ji} d_{ji} (\tau_j - s_i)]\}\}$
Min-Th: Minimizing & balancing thermal hot spots	Minimize H; $H = max\{Q_p; p = 1...m$, for a system of m cores$\}$ where: $Q_p = \sum_{T_i \in T} \{x_{ip} \sum_{f_k} (q_{ik} y_{ik})\}$
Bal-En: Balancing energy consumption	Minimize EN_{max}; $EN_{max} = max\{EN_p; p = 1...m$, for a system of m cores$\}$ where: $EN_p = \sum_{T_i \in T} \{x_{ip} \sum_{f_k} (e_{ik} y_{ik})\}$
Min-En: Minimizing total energy	Minimize EN_{total}; $EN_{total} = \{\sum_{T_i \in T} \sum_{f_k} e_{ik} y_{ik}\} + I_{total}$; $I_{total} = \sum_{p \in PU} \{ \sum_{i,j \in T, i \neq j} x_{ip} x_{jp} m_{ij} \ \ idle(s_j - \tau_i)\}$

Table 3.2: ILP Formulation for Min-Th&Sp

Minimize $H + G$; $H = max\{Q_p; p = 1...m$, for a system of m cores$\}$ where: $Q_p = \sum_{T_i \in T} \{x_{ip} \sum_{f_k} (y_{ik} q_{ik})\}$ $G = \sum_{p,r \in PU, p \neq r} \{n_{pr} \{ \sum_{i,j \in T, i \neq j} x_{ip} x_{jr} [p_{ij} d_{ij} (\tau_i - s_j) + p_{ji} d_{ji} (\tau_j - s_i)]\}\}$ Subject to constraints:	
(a) $\forall T_i : \sum_p x_{ip} = 1$	Each task is assigned to only one PU
(b) $\forall T_i : \sum_k y_{ik} = 1$	Each task runs at only one V/f level
(c) $\tau_i = s_i + t_i$	Execution finish time for T_i
(d) $s_i \geq max_{E_{ji} \in E}\{\tau_j\}$	Task precedence
(e) $\tau_i \leq D_i$	Deadlines for all sink nodes
(f) $s_i \geq \tau_j;\ if\ p_{ji} = 1$	Precedence for tasks on the same core
(g) $p_{ij} + p_{ji} = 1$; if $x_{ip} = x_{jp} = 1$	If T_i and T_j are scheduled on the same core, either T_i precedes T_j, or vice versa

(i.e., same peak temperature) for a second. Or, two systems can have the same average temperature but very different thermal profiles. Having a time-based metric addresses such differences.

To compute the time spent above the threshold for each task (q_{ik}), we perform thermal simulations. The temperature profile for a job depends on the allocation and the floorplan of the MPSoC in addition to the individual power characteristics of the job. Our ILP-based static optimization is designed for a relatively small-sized task graph with long tasks. Therefore, temperature during a task's execution is strongly affected by the execution time of the task. We initially assume the time spent above the threshold for each task is equal to the execution time (WCET) of the task ($q_{ik} = WCET_i$). We solve the ILP (`Min-Th&Sp`) for the given task graph, maintaining the deadlines and precedence constraints among jobs. We next perform thermal simulation, and record the time spent above the threshold temperature for each task. Afterwards, we insert these new q_{ik} estimates in the ILP, and solve the ILP again to get the final scheduling results. We set the threshold temperature to 85^oC in our simulations, as 85^oC is considered a high temperature for many processors [58].

The iterative stage included in the method is for refining the per-task thermal

Table 3.3: Variables Used in the ILP

x_{ip}:	Set of 1-0 variables s.t.* $x_{ip} = 1$ iff T_i is assigned to PU_p
q_{ik}:	Time spent above threshold temperature while running T_i at f_k
t_i:	WCET of T_i considering the voltage setting
s_i:	Execution start time for T_i
τ_i:	Execution finish time for T_i
p_{ij}:	Set of 1-0 variables s.t. $p_{ij} = 1$ iff T_i starts before T_j
n_{pr}:	Set of 1-0 variables s.t. $n_{pr} = 1$ iff p and r are adjacent cores
d_{ij}:	Set of 1-0 variables s.t. $d_{ij} = 1$ iff $\tau_i \geq s_j$
y_{ik}:	Set of 1-0 variables s.t. $y_{ik} = 1$ iff T_i runs at speed f_k
m_{ij}:	Set of 1-0 variables s.t. $m_{ij} = 1$ iff T_j immediately follows T_i
	* *s.t.: such that*

behavior estimates (i.e., q_{ik} values). Having accurate thermal estimates is essential for achieving ILP results close to the optimal schedule. To verify the accuracy of the iterative thermal estimation method, we perform simulations on 5 randomly generated task sets, each set containing 10 tasks. First, we solve the ILP using the thermal estimates that are obtained as described above. We then simulate the temperature response for the schedule determined by the ILP, and record the time spent above the threshold. Iterating on this loop several times (i.e., getting new q_{ik} estimates from thermal simulation, feeding them into the ILP, and solving the ILP again with the new q_{ik} values), we observe that the error in temperature estimation stays below 5%.

In the computation of the Q_p component of H, y_{ik} is an integer variable that is 1 *iff* task T_i is scheduled to run at frequency f_k, and q_{ik} is the thermal estimate for T_i at frequency f_k. The ILP formulation we provide here for `Min-Th&Sp` can be applied to systems that have dynamic power management (DPM) or dynamic voltage scaling (DVS). For the cases without power management or for DPM only, there is only one voltage setting, so $y_{i1} = 1$. For DPM, the q_{ik} estimates should be derived through simulations with DPM, as putting the cores into sleep state is expected to affect the thermal behavior. For DVS cases, we assume that each task will run on a fixed frequency. We then need to evaluate the thermal profile of each task for all frequency/voltage levels.

In the second part of the objective function of `Min-Th&Sp` (G in Table 3.2), we compute the total *overlap* in the schedule. We use two additional variables while formulating the overlap: n_{pr} is an integer variable which is equal to 1 only if cores p and r are adjacent to each other in the floorplan; d_{ij} is an integer variable that is equal to 1 only if the completion time of T_i is greater than the start time of T_j. The x variables

check if two tasks are scheduled on neighbor units. The product $p_{ij}.d_{ij}$ is equal to 1 if T_i precedes T_j and if T_j starts before T_i finishes. So, when $p_{ij} \cdot d_{ij} = 1$, there is an overlap of T_i and T_j. The difference $t_i - s_j$ quantifies the duration of the overlap.

Table 3.2 also demonstrates the ILP constraints that guarantee the deadlines and task precedence. The x integer variables defined in (a) assure that each task is assigned to only one core, and (b) shows that each task runs at a fixed voltage setting. Constraint (c) computes the finish time of tasks to guide the precedence and deadline constraints. We use two sets of precedence constraints. The first set, (d), makes sure the dependences between tasks are satisfied, so that a consumer task does not start before all its producer tasks complete. In addition to this, if several tasks are scheduled on the same core, a task can only start after the previously scheduled tasks are completed (f). We define the p variables in (g) to help defining the constraints in (f). Constraint (e) ensures that all tasks with deadlines finish before their deadlines.

We next provide the details for the other ILP formulations presented in Table 3.1, and point out their differences with `Min-Th&Sp`.

Minimizing and balancing the thermal hot spots: `Min-Th` minimizes the maximum time spent above threshold temperature for each core in order to minimize and balance the thermal hot spots. In the ILP formulation for `Min-Th`, the second part of the objective function is eliminated (i.e., $G = 0$) as this ILP does not consider spatial gradients. The rest of the formulation is the same.

Energy balancing: The ILP formulation for `Bal-En` has the overlap set to zero ($G = 0$), and the temperature variable q_{ik} in `Min-Th&Sp` is replaced with e_{ik}, which is the energy per task for running T_i at frequency f_k. y_{ik} is an integer variable that is 1 *iff* T_i runs at frequency f_k. Summing the expression $e_{ik}y_{ik}$ computes the energy consumed per task.

Minimizing energy: The ILP for `Min-En` is applicable to systems with DPM, dynamic voltage scaling DVS or both. For systems with DPM only, the ILP is solved for only the default frequency of the system. For DVS, the $\sum_{f_k} e_{ik}y_{ik}$ term computes the energy per task for the frequency level task T_i is assigned. In that case, the timing parameters (e.g., t_i, s_i, etc. in Table 3.3) are computed considering the voltage settings of tasks.

While computing the total energy EN_{total}, we consider the energy consumed during all active and idle periods. In order to compute the length of idle time slots, we

define an integer variable, m_{ij}, which is 1 iff task T_i starts before T_j, and there is no other task whose start time is between s_i and s_j. Minimizing the total energy involves minimizing the energy consumption during idle time slots as well. I_{total} is the energy spent during all idle times, and the *idle(y)* function computes the energy for individual idle time slots. For example, when we apply a fixed timeout DPM strategy which puts cores into sleep state if the idle time is longer than a given timeout ($t_{timeout}$), the energy during idle time slot S can be computed as below. Here, $e_{penalty}$ and $t_{penalty}$ are the energy and time overhead for switching into and out of the sleep state, respectively.

$$idle(S) = e_{penalty} + e_{slp}(S - t_{penalty}) \;\; if \;\; S \geq t_{timeout} \tag{3.1}$$

$$idle(S) = e_{idle} \cdot S \;\; if \;\; S < t_{timeout} \tag{3.2}$$

3.1.1 Linearization

We have described the detailed formulations of the ILPs for `Min-Th&Sp`, `Min-Th`, `Min-En` and `Bal-En`. These ILP formulations include multiple nonlinear equations. Solutions to similar nonlinear problems have been presented in [51] and [74]. Such problems can be solved by ILP solvers after linearization using standard techniques [10].

First, when two integer variables are multiplied, we introduce a third variable instead of the product, and create additional constraints to define this variable. Assuming variables x_{ip} and x_{jr} are multiplied, we add a third 0-1 variable X_{ipjr} in the formulation with the following constraints:

$$x_{ip} + x_{jr} - X_{ipjr} \leq 1 \tag{3.3}$$

$$-x_{ip} - x_{jr} + 2X_{ipjr} \leq 0 \tag{3.4}$$

When multiplying binary (1-0) variables with integer values, such as $(p_{ij} \cdot s_i)$, we use the following linearization, where D is a suitably large bound for the variables. We define a new variable r_{ij} such that $r_{ij} = p_{ij} \cdot s_i$. The following constraints satisfy the multiplication:

$$r_{ij} - D \cdot p_{ij} \leq 0 \tag{3.5}$$

$$-s_i + r_{ij} \leq 0 \tag{3.6}$$

$$s_i - r_{ij} + D \cdot p_{ij} \leq D \tag{3.7}$$

To linearize the step function introduced by the d_{ij} variables, we use Equation 3.8. The multiplications in this equation are linearized using the methods previously discussed.

$$d_{ij}(\tau_i - s_j) + (1 - d_{ij})(-\tau_i + s_j) \geq 0 \tag{3.8}$$

Though converting the nonlinear problem to a linear one is possible through these techniques, including integer variables in the formulation causes the problem size to grow exponentially as the number of tasks increase. As a result, it may become impractical to solve the scheduling problem for many tasks. Solving scheduling problems using LP-relaxation and randomized rounding has been discussed in previous work (e.g., [69]). For large task sets, ILPs can be solved using LP relaxation as an approximation method, where the relaxation eliminates the integer variables and computes fractions instead. These fractions are then converted back to integral values using randomized rounding. For many NP-hard problems, randomized rounding is shown to yield the best approximation known by any polynomial time algorithm [69]. It should be noted that additional techniques may be required to guarantee the feasibility of the solution of the approximation algorithm. Solving the ILP using an approximation algorithm is beyond the focus of this work.

3.2 Experimental Methodology

Our experimental results are based on the UltraSPARC T1 processor [45], which contains 8 cores and memory, communication and I/O units. This MPSoC has been manufactured in 90 nm process technology. The processors are in-order execution cores and have multithreading capability. In each core, 4 threads share an integer pipeline. Every two cores share an L2-cache and the cores communicate through shared memory. The power distribution among the units and relative sizes of each unit on the chip are provided in Table 3.4. The power data are updated values for those reported in [45], and they include the leakage estimates. UltraSPARC T1 (floorplan shown in [45]) runs a multilevel queuing scheduler with basic load balancing capabilities as default.

We leverage Continuous System Telemetry Harness (CSTH)) [27] to gather detailed workload characteristics of real applications. CSTH collects and analyzes real time

Table 3.4: Power and Area Distributions of the Units

Component Type	Power (%)	Area (%)
Cores	65.27	37.66
Caches	25.50	50.69
Crossbar	6.01	5.84
Other	3.22	5.81

data from hardware sensors, (e.g., currents, voltages and temperatures) as well as software variables (e.g., performance metrics, memory accesses, etc.). CSTH runs as a part of the existing system software stack; therefore, the data processing does not introduce additional overhead.

We need to have a power consumption trace for each unit on the die to perform the thermal evaluation of the scheduling techniques. If we know when each unit is active or idle, we can estimate the instantaneous power consumption using the average power values. For SPARC cores, the peak power consumption is very close to the average power values [45]. Therefore, for cores, our goal is to determine when each core is active.

We sample the utilization percentage for each hardware thread at every second using `mpstat` [48]. `mpstat` provides the distribution of user, kernel and idle times. We record the utilization traces for half an hour for each benchmark. To determine the active/idle time slots of cores more accurately, we use the kernel probes in `DTrace` [48] for recording the length of user and kernel threads. `DTrace` is a comprehensive dynamic tracing framework for Solaris. It should be noted that the lengths of threads we measure may not correspond to the overall execution time for a thread. For example, a thread might run for several minutes, but due to context switches, the continuous execution slices are of shorter length. Based on the length of the threads and the utilization traces obtained using `mpstat`, we reconstruct the workload trace.

We use a set of benchmarks, which are grouped in four categories: 1) Web server, 2) Database applications, 3) Commonly used integer benchmarks, 4) Multimedia benchmarks. For generating web server workload, we use SLAMD [60], which is a distributed application for load generation. The number of clients and threads can be tuned, allowing for simulating various workload intensity. We run SLAMD with one client, and 20 and 40 threads per client to achieve medium and high utilization ratios respectively.

To generate workload for database applications, we use MySQL, and test it with a multithreaded benchmarking tool, `sysbench`, with various table sizes and number of

Table 3.5: Workload Characteristics

Benchmark	Core Util. (%)	Thread Length (ms)	L2 I Miss	L2 D Miss	FP inst.	Thermal HS	Thermal TC
Web-med	53.12	134	12.9	167.7	31.2	X	X
Web-high	92.87	268	67.6	288.7	31.2	X	
Database	17.75	268	6.5	102.3	5.9		
Web & DB	75.12	536	21.5	115.3	24.1	X	X
gcc	15.25	268	31.7	96.2	18.1		
gzip	9	536	2	57	0.2		
MPlayer	6.5	268	9.6	136	1		
MPlayer &Web	26.62	134	9.1	66.8	29.9		X

threads. Using `sysbench` we create a table with 1 million rows and 100 threads to access the database. For servers, combination of web and database applications are very commonly observed; therefore we include the `Web&DB` application in our benchmark set as well. We also run compiler (`gcc`) and compression/decompression (`gzip`). For the multimedia benchmarks, we run MPlayer (integer) with a 640x272 sample video file. While simulating gcc and gzip, we run 6 simultaneous copies of the application to increase the system utilization. While simulating MPlayer, we use 4 instances of the video application for the same reason. We do not increase the number of applications further as these applications tend to become I/O and memory bound.

We summarize the details for our benchmarks in Table 3.5. In the table, we demonstrate the system utilization, which is averaged over all cores and throughout the execution. We also provide the maximum thread lengths measured. We gather additional information for each benchmark using `cpustat`, such as cache misses and floating point instructions (See Table 3.5). We use these characteristics to model the power consumption of the crossbar and floating point unit. We report the L2 cache misses, as these give an idea about how frequently the crossbar is accessed. The memory and floating point statistics are per 100K instructions. In the table we compare the thermal profiles of the benchmarks as well. The "X" marks show the benchmarks that have high percentage (over 25%) of hot spots (HS) and high-magnitude thermal cycles (TC). The classification for spatial gradient profiles are similar to that of hot spots, so we report only one of them here. The benchmarks without "X" are prone to hot spots

and variations for a low to medium percentage of time.

We implement a simulator to fairly compare different scheduling techniques. In our simulation, we take representative traces collected at runtime for each workload category. For evaluating the static approaches, based on these traces, we design task graphs consisting of 10 tasks for each benchmark that matched the utilization and task length characteristics. We simulate task graphs with and without task dependences.

We solve the ILPs in our static method using lp_solve [47], which is able to solve an ILP for a set of 10 tasks in 2 hours. In the first step of the simulator, the scheduler is given a list of jobs and their start times, which is provided by the ILP solution. The scheduler then applies a fixed scheduling strategy based on the ILP results. Thus, the performance overhead of this method during runtime is minimal.

In the next step of the simulator, power values are derived based on each unit's execution profile. Dynamic power management or voltage scaling is also applied at this stage, depending on the policy simulated. For cores, we use average power values for the active and idle states for UltraSPARC T1. In the average case, the ratio between active and idle state power is 7.4. We estimate the dynamic power at the lower voltage levels based on the relationship between power, frequency and voltage (i.e., $P \propto f * V^2$). We assume two built-in voltage/frequency settings in our simulations. To account for the leakage power, we apply the second-order polynomial model proposed in [66]. This model computes the change in leakage power for the given differential temperature and voltage values. We determine the coefficients in the model empirically to match the normalized leakage values in [66]. This second-order model is shown to match closely with measurements. As we know the amount of leakage at the default voltage level for each core, we scale it based on this model for each voltage level, taking both the temperature and voltage change into consideration. We use a sleep state power of 0.02 Watts, which is estimated based on sleep power of similar cores. For DPM, we implement a fixed timeout policy [4] with timeout set to 100ms. In addition, we investigate a combined policy of DPM and dynamic voltage scaling (DVS). The hybrid DPM/DVS policy selects the lowest frequency possible for each task considering the deadline constraints, and shuts down the cores based on the fixed timeout policy. For the crossbar, we use a simple power model, where the power consumption scales according to how many cores are active and their L2 access characteristics derived from traces in Table 3.5.

The next step is to obtain the temperature distributions using a thermal simu-

lator. We utilize HotSpot version 4.2 [59] as the thermal modeling tool, and modify it accordingly our MPSoC. The thermal package characterization is based on the package properties of the UltraSPARC T1. We perform the thermal simulations using a sampling interval of 10 ms, which provides a good precision. We initialize HotSpot simulations with the steady state temperature values.

3.3 Experimental Results

This section compares various static optimization techniques by contrasting their efficiency of reducing thermal hot spots, spatial gradients, and temporal fluctuations (i.e., thermal cycles). We show results for systems with DPM and DVS strategies to demonstrate how the schedulers perform when the system has power management capabilities. In addition to all the static optimization techniques discussed in this chapter, we also show results for Dynamic Load Balancing (DLB), which is a commonly used policy in multicore schedulers. DLB, in this implementation, sends workload to the least utilized core at runtime to balance the load. It does not perform any thermal management.

The hot spot results show the percentage of time spent above 85^oC, which is considered a critical temperature for our system. The recommended maximum die temperatures for Intel 1.5 GHz Pentium 4 processor and AMD 1.2 GHz Athlon processor are 72^oC and 95^oC, respectively [58]. The spatial gradient results summarize the percentage of time gradients above 15^oC are observed. Device delay is correlated with on-resistance, which increases with temperature. Gradients of even $15 - 20^oC$ start causing clock skew and delay issues [1]. The spatial gradient distribution is calculated by evaluating the temperature difference between hottest and coolest cores at each sampling interval.

We report the temporal fluctuations of magnitude above 20^oC for only the cases with DPM and DVS/DPM, because going into sleep state causes large magnitude of variations in temperature. We do not provide thermal cycling results for the the case of no power management due to the lack of significant temporal variations (hence the results are not available for this case in Table 3.6). The number of cycles to failure can be approximated using Coffin-Manson model [34]. For example, if we compute the failure rate for metallic structures, assuming the same frequency of cycles, when ΔT increases from 10 to 20^oC, failures happen 16 times more frequently. Thermal cycling results are obtained by computing the ΔT over a sliding window and averaging the ΔTs of all cores.

We next provide an extensive comparison of the ILP based techniques. We refer

Table 3.6: Summary of Experimental Results

	Thermal Hot Spots($\% > 85^oC$)			Thermal Cycles ($\% > 20^oC$)			Spatial Gradients ($\% > 15^oC$)		
Benchmark	DLB	Min-En	Min-Th&Sp	DLB	Min-En	Min-Th&Sp	DLB	Min-En	Min-Th&Sp
	No Power Management								
AVG	21.2	N/A	4.5	N/A	N/A	N/A	9.0	N/A	0.8
	DPM								
Web-med	27.2	8.5	7.5	36.6	21.8	6.5	17.0	9.4	2.1
Web-high	47.5	14.8	12.2	12.2	7.2	1.9	28.7	15.7	1.5
Database	9.7	2.8	0.0	22.3	10.3	3.0	6.2	3.6	1.2
Web&DB	38.4	12.0	10.6	29.5	15.0	3.5	23.8	13.1	1.1
gcc	7.2	2.5	0.0	20.3	12.3	1.9	4.6	2.5	0.0
gzip	4.4	1.5	0.0	12.0	7.3	1.5	2.8	1.5	0.0
MPlayer	3.0	1.0	0.0	9.5	6.0	1.5	2.6	1.7	0.0
MPlayer&Web	14.4	4.2	0.1	29.2	18.2	3.1	8.7	4.9	1.2
AVG	19.0	5.9	3.8	21.5	12.3	2.9	11.8	6.6	0.9
	DVS & DPM								
Web-med	20.4	4.8	5.1	22.0	10.8	3.5	13.4	7.5	1.2
Web-high	33.4	8.0	8.2	6.8	3.7	1.0	17.4	8.7	1.5
Database	7.3	2.1	0.0	12.9	5.3	2.1	4.0	2.3	1.1
Web&DB	26.1	5.8	6.8	17.1	7.3	2.7	15.6	7.9	1.0
gcc	5.1	1.5	0.0	11.4	6.1	1.9	4.4	2.3	0.0
gzip	3.4	0.8	0.0	7.3	3.8	1.2	2.3	1.2	0.0
MPlayer	2.2	0.6	0.0	5.7	2.9	0.9	2.6	1.6	0.0
MPlayer&Web	10.8	2.6	0.1	17.4	9.2	2.8	7.2	4.5	0.7
AVG	13.6	3.3	2.5	12.6	6.1	2.0	8.4	4.5	0.7

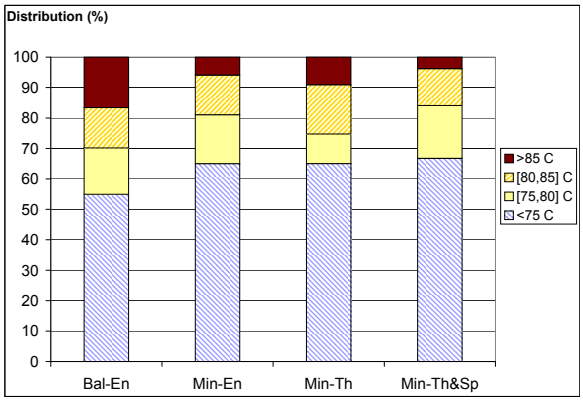

Figure 3.2: Distribution of Thermal Hot Spots (with DPM)

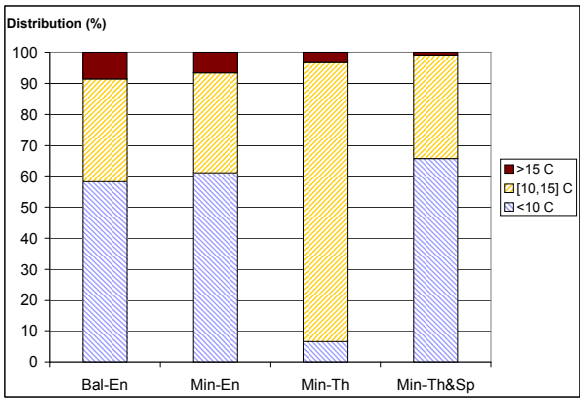

Figure 3.3: Distribution of Spatial Gradients (with DPM)

to our static approach as `Min-Th&Sp`. As discussed in Section 3.1 we formulate the ILP for minimizing thermal hot spots (`Min-Th`), energy balancing (`Bal-En`) and energy minimization (`Min-En`) to compare against our approach. To the best of our knowledge, this is the first time in the literature static MPSoC scheduling techniques are compared extensively to evaluate their thermal behavior.

We first show average results over all the benchmarks. Figure 3.2 demonstrates the percentage of time spent at certain temperature intervals for the case with DPM. The figure shows that `Min-Th&Sp` achieves a higher reduction of hot spots in comparison to the other energy and temperature based ILPs. The reason for this is that, avoiding clustering of workload in neighbor cores reduces the heating on the die, resulting in lower temperatures.

Figure 3.3 shows the distribution of spatial gradients for the average case with

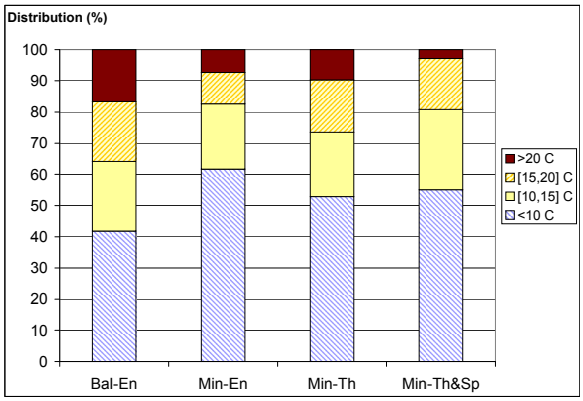

Figure 3.4: Temporal Variations (with DPM)

DPM. In this plot, we can observe how `Min-Th` increases the percentage of high differentials while reducing hot spots. While `Min-Th` reduces the high spatial differentials above 15^oC, we observe a substantial increase in the spatial gradients above 10^oC. In contrast, our method achieves lower and more balanced temperature distribution in the die.

In Figure 3.4, we show how the magnitudes of thermal cycles vary with the scheduling method. We demonstrate the average percentage of time the cores experience temporal variations of certain magnitudes. As can be observed in the figure, `Min-Th&Sp` reduces the thermal cycles of magnitude 20^oC and higher significantly. The temporal fluctuations above 15^oC are reduced in comparison to other static techniques, except for `Min-En`. The cycles above 15^oC (total) occur 17.3% and 19.2% of the time for `Min-Th&Sp` and `Min-En`, respectively. Our formulation targets reducing the frequency of highest magnitude of hot spots and temperature variations, therefore such slight increases with respect to `Min-En` are possible.

In the plots discussed above and also in Table 3.6, we observe that the `Min-Th&Sp` technique successfully reduces hot spots as well as the spatial and temporal fluctuations. Power management (see DPM and DVS&DPM results for `Min-En` in Table 3.6) reduces the hot spots to some extent, but it cannot eliminate them effectively. Moreover, applying power management creates thermal cycles and larger spatial gradients due to the considerable decrease of power in sleep state. For example, `Bal-En` has high magnitude of cycles for 16% of the time (for DPM). `Min-En` reduces this percentage to about 7%. This reduction is due to the decrease in high temperatures. `Min-Th&Sp` can further decrease the frequency of cycles to less than 3%. We also observe that combining DVS with DPM reduces both high temperatures and temperature variations in comparison to

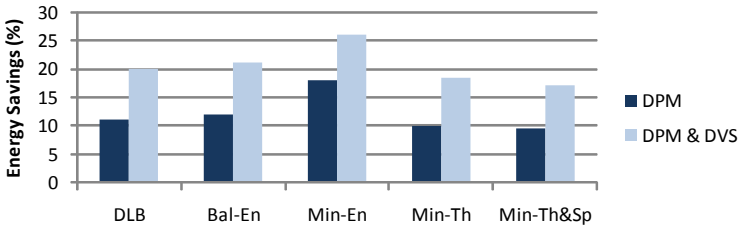

Figure 3.5: Energy Savings with DPM and DVS&DPM

applying only DPM.

The temperature balancing approaches `Min-Th` and `Min-Th&Sp` achieve much lower frequency of spatial gradients in comparison to energy-based techniques. For the cases with DVS&DPM, `Min-Th&Sp` bounds the frequency of spatial gradients to below 1.5% for all benchmarks except Web-high, which has over 90% utilization and a considerably high percentage of thermal hot spots.

`Min-Th&Sp` achieves more dramatic reductions in hot spots and gradients for benchmarks with lower system utilization (e.g., gcc and gzip), since the optimization method has more freedom to distribute the workload across the chip. As utilization increases (e.g., Web & DB and Web-high), we observe an increasing percentage of hot spots; however, the thermal cycles decrease as the system does not go into sleep state as often.

In Figure 3.5, we show the energy savings of DLB and static optimization techniques when they are integrated with DPM and DVS&DPM. The results are averaged over all the workloads, and normalized with respect to running DLB without any power management. As expected, `Min-En` achieves the highest savings: 18% with DPM, and 26% with DVS&DPM. Both `Min-Th` and `Min-Th&Sp` sacrifice energy savings for reducing hot spots. These two policies distribute workload more evenly among the cores in comparison to other policies; hence, they reduce the amount of continuous idle time slots that are utilized by DPM's sleep modes. The average energy reduction achieved by our technique, `Min-Th&Sp`, is 9.5% with DPM and 16% with DVS&DPM. Thus, we can still reduce the energy consumption to a great extent while minimizing the hot spots and gradients at the same time.

We have seen that our technique, `Min-Th&Sp`, outperforms other energy and temperature based ILPs in terms of reducing both hot spots and temperature gradients.

Minimizing energy (`Min-En`) reduces the hot spots due to the decrease in power, and manages to reduce gradients to some extent. However, by considering thermal profiles of tasks and the location of cores on the chip, `Min-Th&Sp` can achieve lower and more even temperature profiles.

3.4 Summary

In this chapter, we have proposed a static ILP-based job scheduling optimization for MPSoCs. Static optimization sets a baseline for dynamic techniques, and it can be utilized for systems with known workloads, such as some embedded systems. The proposed static scheduler minimizes the hot spots, as well as the temporal and spatial temperature variations, while meeting the timing and precedence constraints of the workload. We also formulate ILPs for minimizing energy, balancing energy, and minimizing hot spots (without considering gradients), and provide an extensive comparison. We demonstrate that our static temperature-aware scheduling method outperforms the other energy or temperature based optimal approaches in terms of reducing both hot spots and gradients. For example, the ILP for minimizing energy achieves 18% energy savings with dynamic power management (DPM). However, it cannot prevent hot spots and causes larger thermal variations due to the ultra-low power sleep states. Our technique reduces the frequency of hot spots by 35%, spatial gradients by 85% and thermal cycles by 61% in comparison to the ILP for minimizing energy. At the same time, we still achieve close to 10% savings in energy.

The results of this chapter highlight the following important points:

- Energy management is not sufficient to achieve safe and stable temperature profiles, even though techniques such as dynamic power management reduce the average temperature. DPM prefers to cluster the workload to extend the idle time slots and to utilize these longer idle periods for saving energy. Moreover, as DPM does not consider the location of the cores on the die and does not try to spread out the heat across the die, it can create spatial gradients.

- Putting cores into sleep state can potentially increase the thermal cycles and degrade reliability. Designing thermal management policies that address thermal cycles is crucial for achieving high reliability while saving energy.

- Temperature-aware optimization achieves dramatic reduction of hot spots and thermal variations over dynamic load balancing, which is a commonly used technique for multicore scheduling. The guidelines of the static optimization (i.e., balancing the temperature and spreading out the heat dissipation across the die to reduce the variations) should be considered while designing dynamic management policies.

The text of Chapter 3 is in part a reprint of the material from the paper, *Ayse K. Coskun, Tajana Simunic Rosing, Keith Whisnant and Kenny Gross, "Static and Dynamic Temperature-Aware Scheduling for Multiprocessor SoCs"*, in IEEE Transactions on VLSI, September 2008. The dissertation author was the primary researcher and author, and the co-authors involved in the publication [21] directed, supervised, and assisted in the research which forms the basis for that material.

Chapter 4

Low-Overhead Dynamic Temperature Management

Workload characteristics typically vary dynamically during execution, making it hard to predict the workload during system design. Thus, dynamic management of temperature is required to achieve higher system reliability and to reduce the design challenges caused by hot spots and temperature variations. To avoid high performance impact, temperature-aware scheduling has to be fast and easy to implement.

This chapter first presents OS-level temperature-aware scheduling with negligible performance overhead. This technique, called *Adaptive-Random*, mitigates the thermal hot spots and large temperature variations by taking into account the temperature measurements of the MPSoC and adapting to changes. Current chips typically contain several thermal sensors, and these sensors are read by an infrastructure such as the Continuous System Telemetry Harness (CSTH) [27] without introducing performance cost. When combined with previously introduced reactive thermal management methods such as thread migration [26] and voltage scaling [59], *Adaptive-Random* achieves even lower and more stable thermal profiles while reducing the performance impact of such reactive techniques significantly.

We observe that different power or thermal management policies are most advantageous for particular workload scenarios. For example, while *Adaptive-Random* is successful in balancing temperature, it may not provide as much energy savings as a DPM policy (that utilizes sleep modes for idle cores) for a system with low utilization. On the other hand, DPM may accelerate thermal cycles for a system with highly variant

workload. We propose using *Online Learning* for selecting the management policy with the best fit to the current workload conditions by evaluating both the thermal impact and performance of policies.

Both thermal management policies we introduce in this chapter are cost-effective and can be easily implemented into existing schedulers at the OS level.

4.1 Dynamic Temperature-Aware Job Scheduling

Dynamic thermal management typically controls hot spots by keeping the temperature below a critical threshold. Computation migration and fetch toggling are examples of such techniques [59]. Heat-and-Run performs temperature-aware thread assignment and migration for multicore multithreaded systems [26]. Kumar et al. propose a hybrid method that coordinates clock gating and software thermal management techniques [40]. The multicore thermal management method introduced in [24] combines distributed DVS with process migration. These DVS or thread migration based techniques typically come at a substantial performance cost. In [65], the authors propose dynamic MPSoC temperature-aware scheduling methods that take core temperatures into account while making decisions. Their technique is lower cost in comparison to DVS or migration based techniques; however, temperature variations are not considered during scheduling.

This section first discusses the multiprocessor schedulers in state-of-art operating systems, and then provides the details of two previously introduced reactive thermal management methods that are applicable to MPSoCs. Finally, we explain the proposed low-overhead dynamic temperature-aware scheduling technique, *Adaptive-Random*.

4.1.1 State-of-the-Art Load Balancing Schedulers

Many modern OS schedulers are based on multilevel queuing, which mixes several elements such as priority, round-robin and shortest-job-first scheduling principles. For performance reasons, some amount of load balancing is commonly integrated in the scheduler. In Linux 2.6, each processor in the multiprocessor system has a queue. A task stays in a queue for cache affinity. Tasks are moved to different queues only when the load is unbalanced (i.e., when length of a queue is less than one fourth of another). Solaris migrates threads to other processors when a core becomes overloaded. The thread migra-

tion in Solaris is performed based on giving priority to locality, following the assumption that the threads on nearby cores share the same caches.

In this work, we implement a dynamic load balancing policy (*DLB*) where the scheduler balances the workload by sending workload to the least busy processor at each interval (as discussed in Chapter 3 previously). This dynamic load balancing strategy is in principal similar to load balancing performed by operating systems such as Solaris, which balances the workload in the processors' queues at regular intervals.

4.1.2 Thermal Management Techniques for MPSoCs

Several techniques have been proposed in the literature to control the thermal behavior of MPSoCs. Here we discuss two previously introduced methods for managing temperature. Both of these techniques are reactive, that is they are activated only when a critical temperature is reached.

Dynamic Thread Migration (DTM) is an MPSoC thermal management method that migrates threads from hot processors to cooler ones. For minimizing the performance impact of thread migration, Heat-and-Run proposed loading the cores as much as possible and migrating workload when critical temperature values are observed [26]. In our implementation of this technique, we migrate the thread from the hot processor to the coolest processor available at that moment. The threshold temperature for migration is set at 85^oC, which is considered a high temperature for our system. Similar temperature values are shown as critical for other CPUs as well [58].

Voltage Scaling for Thermal Management (VSTM) performs dynamic voltage and frequency scaling when the temperature reaches the threshold [59]. This technique lowers the temperature on the hot cores by reducing power consumption. In our implementation, we assume two built-in voltage/frequency settings for each core. Normally all jobs run at full speed (f_{max}). If a core reaches the critical temperature (85^oC), the frequency/voltage level of the particular core is reduced to the lower setting (f_{low}) until the current job terminates. In our experiments, we picked f_{low} as two thirds of f_{max}. Lower frequency settings can be used as well; however, this would increase the performance cost.

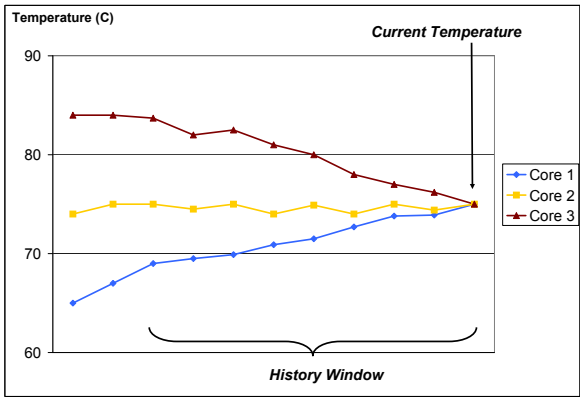

Figure 4.1: Effect of Temperature History

4.1.3 Low-Overhead Temperature Aware Scheduling

One of the goals when designing a dynamic temperature management policy is to have minimal performance overhead. The policy proposed next, *Adaptive-Random*, has negligible overhead in comparison to the existing decision-making process in OS-level multiprocessor schedulers, and it can be implemented in the OS scheduler with minimal changes.

Adaptive-Random updates probabilities of sending workload to cores at each interval based on an analysis of the temperature history on the chip. Taking the history into account provides the ability to allocate workload on units exposed to lower thermal stress or that are on cooler parts of the MPSoC. For example, in Figure 4.1, we see three cores with the same current temperature, but different histories. Assuming all cores are idle, the if we were to make decisions on current temperature only, we would not differentiate among these three cases. *Adaptive-Random* favors *Core-1* due to the lower temperature average in the history window. *Core-1* is indeed a better choice than the others, as the thermal history suggests *Core-1* and its neighbors have been under lower thermal stress. This way, *Adaptive-Random* achieves a better load distribution for performance purposes.

In *Adaptive-Random*, the new probability value for each core is computed using Equation 4.1 at each job arrival. In the equation, P_n is the new probability, P_o is the previous probability, and W is the weight. P_n values saturate at 0 and 1. In order to evaluate the thermal stress on each core, W is computed at regular intervals using a sliding window of temperature history. The thermal constant of our system is on the

order a few hundred milliseconds, so we set the interval and sliding window lengths at 1 second in order to account for the rapid changes in temperature. As we compute only Equation 4.1 at workload arrivals, the computation cost of our technique is negligible. Moreover, we do not have to stall execution. Once the probabilities are updated, the core to run the current job is selected through generating a random number.

$$P_n = P_o \pm W \tag{4.1}$$

The probability values are decremented or incremented by W_{dec} or W_{inc}, depending on whether the temperature has risen above the threshold temperature (T_{thr}), or dropped below a second threshold (T_{low}) respectively. In our simulations we set T_{thr} at 80^oC. We use a threshold lower than 85^oC to prevent hot spots before they occur. In order to avoid allocating workload to cores that have temperatures slightly below 80^oC, we use a second threshold, T_{low}, set to 75^oC in our experiments. We do not increase P_n unless in the last interval the core temperature has dropped below T_{low}. Threshold values lower than 75^oC can introduce performance cost, as a number of cores would be idle until their temperatures are below the threshold. Moreover, setting the second threshold too low increases the temperature swing, which may accelerate the thermal cycling. Increasing the threshold value closer to 80^oC reduces its effect. For applying our technique to different systems, different threshold values can be selected following the same principles depending on the system and workload characteristics. There are three cases we consider while adjusting probability values:

1. If there are processors that have exceeded T_{thr} in the past interval, their P_n values are set to 0.

2. When the temperature of a core is between T_{thr} and T_{low}, no action is taken.

3. For cores that are below the second threshold T_{low} in the last interval, we increase their P_n by W_{inc} (see Eqn. 4.2). While calculating W_{inc}, we evaluate Av_{thr}, which is the average temperature below T_{thr} divided by T_{thr}. This way, if a core is cooler than another, its W_{inc} is greater. We select the β value as 0.1 empirically. We simulate β values between 0 and 0.5 at incrementation steps of 0.05, and select the β with the best average case results for reducing hot spots and variations. For different MPSoCs, similar studies can be carried out to select the best β value.

$$W_{inc} \quad = \beta / Av_{thr} \tag{4.2}$$

This scheduling policy can be implemented on a real system easily with very low overhead. Collecting the thermal data and computation of weights do not impose noticeable performance impact. The random number generator can be implemented through a linear-feedback shift register (LFSR), which often already exists on the chip for test purposes. The rest of the computations (i.e., averages and ratios) are carried out incrementally throughout the execution. Another benefit of this policy is that it achieves better load balancing than making decisions solely on instantaneous temperature. The *Adaptive-Random* policy addresses the issues of maintaining a balanced and low temperature profile as well as distributing the thermal stress to cores as evenly as possible throughout system lifetime.

4.2 Thermal Management Using Online Learning

A number of strategies exist to manage temperature, reduce power consumption or to perform task allocation in a temperature or power-aware manner. The policies proposed in the literature have different optimization goals; thus, their advantages vary in terms of saving power, achieving better temperature profiles or increasing performance. For example, DPM can reduce the thermal hot spots while saving power. However, especially when there are frequent workload arrivals, it can significantly increase thermal cycling (as discussed in Chapter 2). Migrating threads upon reaching a critical temperature achieves significant reduction in hot spots. On the other hand, this strategy does not balance the workload across the chip.

This section proposes a novel technique to adapt the thermal management policy to the current workload characteristics. The online learning framework we use, which is based on the *switching experts problem* introduced in [25], selects the best policy for the current system dynamics among a given set of policies after a number of evaluation steps.

In the switching experts problem, there are N *expert* policies. Also, a set of M higher level experts, called *specialists* are defined. A specialist is a higher level policy that determines which expert should run for the next interval. For example, one specialist can choose to run DPM for all intervals, while another one can select a different expert

depending on whether the system has high, medium or low utilization. The specialists are evaluated regularly based on their impact on temperature and performance. The specialist with the highest evaluation score is selected at every interval. We then apply the expert policy determined by this selected specialist for the next interval.

Another method for selecting experts is using insomniac algorithms, where there is a master algorithm that evaluates a given set of experts by comparing the performance of each expert to that of the *best* expert—hence, such algorithms do not utilize specialists. The online learning technique proposed in [23] for DVS is an example of insomniac learning. However for thermal management, evaluating each expert at every interval is infeasible. Temperature of a unit is dependent on the instant power consumption of that unit, as well as the recent temperature history and the power consumption of neighboring units. Therefore, evaluating each expert's thermal behavior and performance accurately would require running each expert on a separate system in parallel. This is obviously an extremely inefficient approach. Thus, we utilize the *specialist* approach proposed for the switching experts problem in [25]. Our algorithm evaluates a subset of *active* specialists at each iteration.

The pseudo-code for our algorithm is provided in Table 4.1. In this technique, at any given time, only one of the experts is active. The decision to switch to another expert (or continue with the current one) is performed at every interval by the specialist currently responsible for decision making. We maintain weight vectors for the specialists, which get updated at every interval based on the observed *loss*. Loss is a non-negative value demonstrating how well a policy performs in terms of the given objective. At every decision point, the specialist with the highest weight factor is selected by the learning algorithm. This way, our technique guarantees converging to the best available policy for the current workload.

Table 4.1: Pseudo-Code for the Online Learning Algorithm

Initialize $w_i = 1/M$ for $i = \{1, 2, ...M\}$
Do for $t = 1, 2, ..., U$
1. Pick the specialist with the highest w_i, and run the expert policy determined by that specialist
2. Compute the loss function for the last interval (L_t)
3. Update the weights for the specialists associated with the active expert:
   $w_i^{new} = w_i^{old} e^{-n \cdot L_t}$ if active
   $w_i^{new} = w_i^{old}$ otherwise

Our *Online Learning* algorithm maintains a weight vector for all the M specialists, $w =< w_1, w_2, ...w_M >$. Each weight w_i represents the suitability of the specialist to the current workload characteristics. We initialize the w_i values by assigning equal weights, $w_i = 1/M$. As shown in the pseudocode, the weights are updated based on L_t, the loss observed during the last interval. At each iteration, we only update the weights of the specialists that are associated with the active ground expert. For example, if the active expert policy has been *DPM* for the last interval, we only update the weights of the specialists that would have selected DPM for that interval.

Equation 4.3 shows the update function. We use an exponential function to update the weights as in [30]. L_t is the total loss computed during the last interval $[t-1, t)$ and n is the learning rate. Selection guidelines for n are explained in detail in [30]. In our experiments, we set $n = 0.75$. As the new weight, w_i^{new}, depends on the previous weight, w_i^{old}, weight update equation contains the history of updates.

$$w_i^{new} = w_i^{old} e^{-n \cdot L_t} \tag{4.3}$$

The loss function takes both temperature and performance characteristics into account. For evaluating the reliability impact of hot spots, observing only the peak or average temperature does not provide a good intuition of the thermal behavior. For this reason, we use the "time spent above temperature threshold" metric (t_{HS}) to capture the impact of hot spots. For thermal cycles, on each core we compute the percentage of time that cycles larger than a given ΔT value are observed (t_{TC}). Similarly, for spatial gradients, we calculate the time during which large gradients occur (t_{SP}).

Table 4.2: Loss Function

Category	Amount of Loss
Hot Spots	t_{HS} (if $t_{HS} > 0$); 0 (ow*) *$otherwise$
Thermal Cycles	t_{TC} (if $t_{TC} > 0$); 0 (ow)
Spatial Gradients	t_{SP} (if $t_{SP} > 0$); 0 (ow)
Performance	$(LA_c - LA_t)$ (if $LA_c > LA_t$); 0 (ow)

The components of the loss function are provided in Table 4.2. To compute the total loss, we normalize each term and then sum all the terms. We use the "Load Average" metric (i.e., LA in Table 4.2) to evaluate the performance cost. We do not use metrics such as IPC or CPI since they are application dependent, and it is not possible to set a threshold value to compute the performance loss. Load average is the sum of

run queue length and number of jobs currently running. Therefore, if this number is low (i.e., typically below 3 or 5, depending on the system), the response time of the system is fast. If this number is getting higher, it means that the performance is getting worse. LA_c and LA_t are the load average for the last interval and the threshold load average, respectively.

Convergence Bound:

The conventional solutions to the switching experts problem require evaluating each of the exponentially many specialists at every iteration (such as in [30], which is an insomniac learning approach). Provided that U (number of iterations in the sequence) and k (number of intervals) are known, if we could keep a weight for all possible exponentially many specialists, the total loss with respect to the best specialist is upper bounded by $k\ lnN + (k-1)ln(U/k)$ [25], where N is the number of experts. This bound is a very tight bound for this problem; however, this algorithm is computationally very costly. The switching experts framework proposed in [25] overcomes this problem by constructing *specialists* for each expert and interval, and by evaluating only the *active* specialists at each iteration. The bound achieved by the specialist algorithm is $k(lnU + o(lnU))$ larger than the bound above. As $N << U$, this bound gives a convergence rate of $O(k\ lnU/U)$.

4.3 Results

The experimental results in this chapter are based on the UltraSPARC T1 processor with characteristics discussed in Chapter 3. Power model, workload characteristics, and thermal modeling framework have been explained in detail in Section 3.2. To evaluate the performance impact, we compute the average delay in the completion time of jobs with respect to the baseline case of load balancing. In thread migration, we assume each migration takes 200ms, according to the results provided in [9]. All the results regarding the *Adaptive-Random* technique are averaged over a hundred runs in order to obtain statistical convergence.

Figure 4.2 shows the effects of the dynamic techniques in reducing hot spots, and also compares the performance cost. The left axis provides the average percentage of time hot spots (over 85^oC) are observed, for the case without DPM. We show the normalized performance of each policy with respect to the baseline case of load balancing on the right axis. While the aggressive reactive techniques (migration and voltage scaling) achieve reduced percentage of hot spots, their performance cost is very high. When

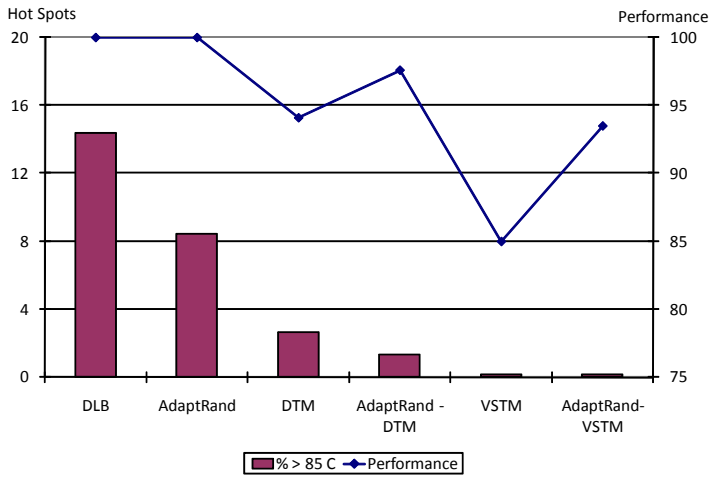

Figure 4.2: Comparison of Hot Spots and Performance Cost

these techniques are combined with *Adaptive-Random*, the hot spots can be further reduced to below 1%, while at the same time the performance degradation is reduced considerably, from 15% to below 7%.

We select a set of expert policies representative of recently proposed power and thermal management approaches for *Online Learning*. This way we cover a variety of trade-off points among temperature, performance and power. The **Default** policy is a performance-oriented policy similar to schedulers in modern operating systems. It tries to allocate threads to the same core they have run previously on to optimize memory accesses. Load balancing is performed if there is congestion. **Dynamic power management (DPM)** turns off idle cores based on a fixed timeout strategy (with timeout set to 200ms as in Chapter 3). **DVS & DPM** applies a dynamic voltage/frequency scaling policy (with three available V/f settings) which reduces the V/f level depending on the utilization observed on each core in the last interval—i.e., it selects the V/f setting to fill in the idle slots as much as possible assuming a linear scaling of execution time with frequency. Idle cores are then turned off using a fixed timeout policy as in DPM. **Migration** moves threads from hot cores to cooler cores when a temperature threshold is exceeded. DPM, DVS & DPM and Migration run together with the Default scheduling policy. We also use the **Adaptive-Random** policy proposed in 4.1 as one of the experts.

As it is practically impossible to maintain a set of specialists covering all possible

segmentations of experts, we use a limited set of specialists. In addition to the specialists that run the same expert all the time (i.e., Default, DPM, DPM&DVS, Migration, Adaptive-Random), we develop specialists that select an expert based on system characteristics to provide faster convergence to the best available policy. These specialists are:

- **Utilization-based:** We observe the average system utilization and select the expert based on the following rules:

  ** High Util.: Migration*

  ** Medium Util.: Adaptive-Random and DVS&DPM*

  ** Low Util.: DPM*

- **Temperature-based:** Based on the thermal profile observed in the last interval, we select the policy for the next interval.

  ** Hot spots or variations: Adaptive-Random*

  ** Otherwise: Default policy*

For our workloads, the set of specialists described above has been sufficient to match a wide range of workload scenarios. It is possible to add other specialists to the *Online Learning* framework for different systems and workloads—especially if there is the need to match the workload characteristics with different expert selection guidelines than those utilized by our specialists.

The loss function has four components that address hot spots, cycles, spatial gradients and performance. Figure 4.3-(a) shows how weighing these components in the loss function changes the frequency each expert is selected. "All equal" assigns equal weights to all components, "P-high" assigns a higher weight to the performance loss, "P-low" assigns higher weight to temperature-related loss and "w/o gradients" compute loss based only on hot spots and performance, without considering gradients. "P-high" selects the policies with minimal performance cost more frequently than others, whereas the "all equal" setting favors Adaptive-Random and Migration more often. DPM and DVS&DPM are typically selected less often than other experts, as they create cycles. In the "w/o gradients" setting, we see a significant increase in the frequencies of selecting DPM and DVS&DPM. In the rest of our evaluation, we use the "all equal" loss function, as we want to minimize all the temperature-induced problems at low performance cost.

In the experimental evaluation, the hot spot results demonstrate the percentage of time spent above 85^oC, which is considered a high temperature for our system. The

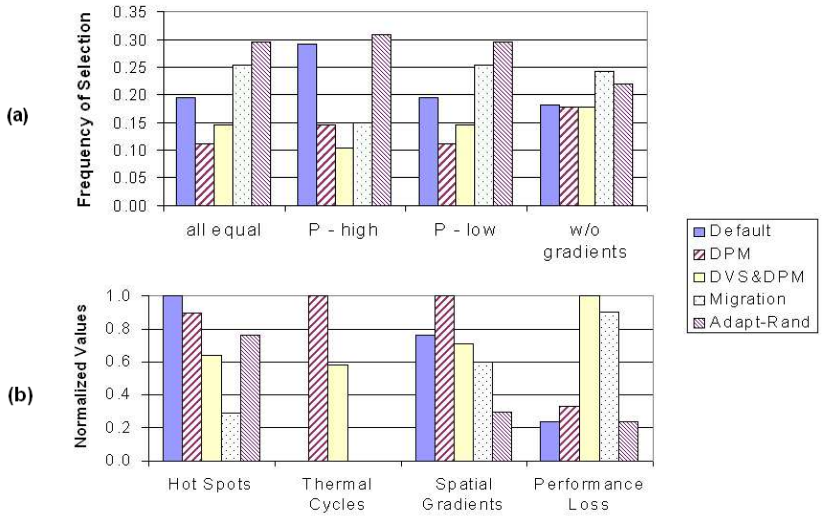

Figure 4.3: (a) Effect of Loss Function on Expert Selection, (b) Evaluation of Expert Strategies

spatial gradient results summarize the percentage of time that gradients above 15^oC occur.. The spatial distribution is calculated by evaluating the temperature difference between hottest and coolest cores at each sampling interval. We report the temporal fluctuations of magnitude above 20^oC. ΔT values we report are computed over a sliding window and averaged over all cores.

In Figure 4.3-(b), we demonstrate how each expert behaves in terms of handling the thermal hot spots and temperature variations, and we compare their performance. These results are averaged over the benchmarks in Table 3.5. We only show thermal cycling results for the experts with DPM, as going into the sleep state causes cycles with high magnitudes. We normalize all values to [0,1] for the sake of comparison. The figure shows the strengths and weaknesses of each expert. For example, `Migration` reduces the thermal hot spots more efficiently than other techniques, however it causes higher performance cost. `DPM` decreases the hot spots and does not have high performance cost, but it causes cycles and gradients.

We run sequences of the benchmarks in Table 3.5 to show how *Online Learning* adapts to varying workload behavior. In our results the sequences are identified by the numbers associated with each benchmark as shown in Table 4.3. For example, the workload sequence A runs Web-med (1) followed by Web-high(2) and so on. In Table 4.3, we compare the efficiency of the *Online Learning (OL)* technique against running each expert alone. For workloads A, B and C, OL reduces the frequency hot

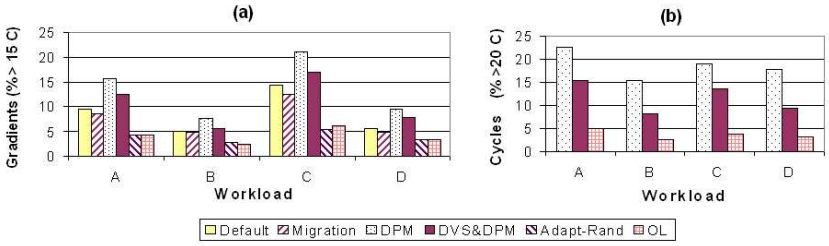

Figure 4.4: (a) Thermal Cycles, (b) Spatial Gradients

spots more than all of the individual experts, as it can combine the advantages of different experts over different execution intervals. For workload D, OL performs almost as good as DPM&DVS. OL reduces the frequency of hot spots by about 20% on average in comparison to DVS&DPM.

Table 4.3: Thermal Hot Spots

W.load	Util (%)	Def.	Migr.	DPM	DVS& DPM	Adapt -Rand	OL
A: 12784	50.8	18.6	11.4	16.9	8.9	13.2	6.5
B: 57843	28.2	8.4	4.2	6.4	2.3	5.4	2.1
C: 14214	69.8	27.8	18.1	21.3	14.9	19.2	9.7
D: 68253	32.3	10.3	5.7	8.0	2.7	6.4	2.9

Figures 4.4 (a) and (b) show how *Online Learning* compares to other methods in terms of reducing temperature variations. Our method reduces the cycles dramatically—i.e., 80% and 68% in comparison to DPM and DVS&DPM, respectively. OL is close to Adaptive-Random in terms of reducing the spatial gradients. We cannot achieve significant reduction of gradients in comparison to Adaptive-Random, because to reduce the hot spots more effectively, our policy sometimes favors other policies.

Figure 4.5 compares the energy consumption and performance of *Online Learning* with the expert policies running alone. Energy and performance results are normalized with respect to the Default policy. DPM and DVS&DPM reduce the energy consumption on average by 13% and 16%, respectively, in comparison to the Default policy. *Online Learning* achieves close to 6% energy savings. As Migration and Adaptive-Random are not integrated with DPM or DVS in these experiments, they do not reduce the energy consumption. In fact, as Migration degrades performance, it slightly increases the overall energy consumption with respect to the Default policy. We compute the performance cost based on the average wait time of jobs on the system. The average wait time of jobs

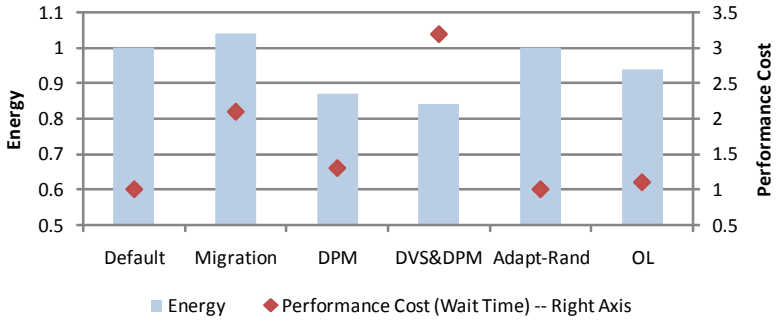

Figure 4.5: Energy and Performance Evaluation

is increased by 2.1 times for Migration, 1.3 times for DPM, and 3.2 times for DPM&DVS. Adaptive-Random does not increase the wait time, and OL only increases it by 1.1 times. Thus, while our technique effectively manages the temperature induced problems at low performance cost, we can also save 6% energy on average.

4.4 Summary

This chapter has presented dynamic thermal management techniques with low performance overhead. First, it has discussed *Adaptive-Random*, which modifies the workload allocation policy based on the temperature history. *Adaptive-Random* is more successful in eliminating hot spots than existing load balancing techniques, and also it provides 50% and 90% reductions in temporal and spatial temperature variations, respectively, with negligible impact on performance. In addition, we have demonstrated that the performance overhead of reactive techniques such as thread migration and voltage scaling can be reduced dramatically when they are combined with the a scheduling policy such as *Adaptive-Random*, while achieving lower and more stable temperatures.

We have also introduced an MPSoC temperature management technique that utilizes *Online Learning* to adapt to dynamically changing workloads. *Online Learning* evaluates a set of "expert" management policies at runtime by taking both thermal behavior and performance into account, and guarantees convergence to the most beneficial policy for the desired performance-temperature trade-off. We have shown that our technique reduces the hot spots, thermal cycles and gradients much more effectively than running a single temperature or power aware policy. For example, a combined DVS&DPM policy reduces the energy consumption by 16% on average, but creates large

thermal variations and does not reduce the hot spots effectively. *Online Learning* reduces the frequency of hot spots by 20% on average in comparison to DVS&DPM, and it is still able to achieve 6% energy savings. Also, *Online Learning* reduces the thermal cycling frequency to below 5% and performs as good as *Adaptive-Random* in minimizing the spatial gradients.

Important conclusions derived in this chapter are:

- As multicore systems get highly complex, management methods to adapt to runtime conditions are required to be able to learn the dynamics and manage the performance, reliability, and energy to meet the system- or workload-specific constraints.

- Dynamic temperature-aware scheduling that makes decisions based on temperature feedback from the system has the ability to substantially reduce the hot spots and thermal variations without a noticeable impact on performance.

- Temperature-aware scheduling is orthogonal to other power management or backup thermal management strategies. In other words, such workload scheduling approaches can be integrated with other management methods to improve the thermal behavior and performance simultaneously.

- The success of power/thermal management or scheduling policies varies depending on how the policy fits the current workload executing on the system. As most systems go through significant changes in workload patterns during execution, it is highly advantageous to analyze the runtime conditions and select an appropriate policy.

The text of Sections 4.1 and 4.3 are in part reprints of the material from the papers, *Ayse K. Coskun, Tajana Simunic Rosing and Keith Whisnant, "Temperature Aware Task Scheduling in MPSoCs"*, in Proceedings of Design Automation and Test in Europe (DATE), 2007, and *Ayse K. Coskun, Tajana Simunic Rosing, Keith Whisnant and Kenny Gross, "Static and Dynamic Temperature-Aware Scheduling for Multiprocessor SoCs"*, in IEEE Transactions on VLSI, September 2008. The dissertation author was the primary researcher and author, and the co-authors involved in the publications [17] and [21] directed, supervised, and assisted in the research which forms the basis for that material.

The text of Sections 4.2 and 4.3 are in part a reprint of the material from the paper, *Ayse K. Coskun, Tajana Simunic Rosing and Kenny Gross, "Temperature Management in Multiprocessor SoCs Using Online Learning"*, in Proceedings of Design Automation Conference (DAC), 2008. The dissertation author was the primary researcher and author, and the co-authors involved in the publication [19] directed, supervised, and assisted in the research which forms the basis for that material.

Chapter 5

Proactive Temperature Balancing

Low-cost dynamic temperature-aware scheduling techniques substantially improve the temperature profiles of MPSoCs. Even though such techniques reduce thermal variations as well as hot spots more effectively in comparison to conventional thermal or power management, they are still reactive in nature, and therefore take action only after undesirable thermal profiles are observed. When we forecast temperature instead of reacting to thermal events, the management policy is able to act before thermal emergencies or imbalances occur. This way, we achieve lower and more stable temperatures with better system performance in comparison to reactive techniques, as we avoid the need to stall or slow down execution upon reaching a threshold temperature.

This chapter proposes a proactive thermal management method for MPSoCs to prevent thermal problems before they occur at a negligible performance cost. In our experiments, we have observed that workloads typically have self-correlation (i.e., current workload behavior depends on the previous actions). As a result, temperature signal is serial correlated as well. Moreover, due to thermal time constants, the temperature changes slowly. Hence, within short time intervals the temperature behavior resembles a stationary time-series signal. This observation enables accurate temperature estimation by regressing on the previous measurements.

The proactive technique proposed in this chapter utilizes an autoregressive moving average (ARMA) model for estimating future temperature accurately based on past temperature measurements. We introduce a novel job scheduling approach, *Proactive Temperature Balancing*, which utilizes the thermal forecast for reducing the hot spots and balancing the temperature among the cores. The ARMA model for prediction may

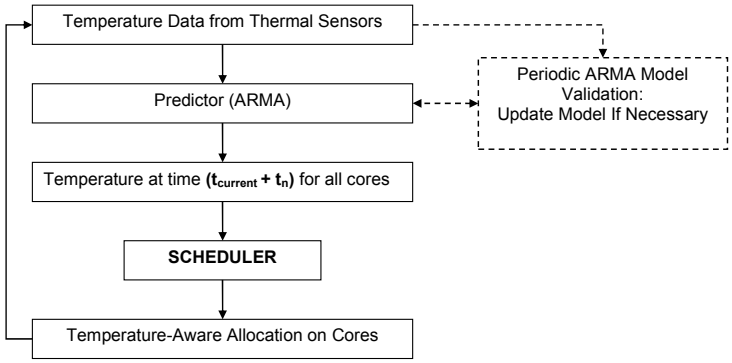

Figure 5.1: Flow Chart of the Proposed Technique

need to be updated if the workload dynamics change substantially. Since the goal is to proactively allocate workload, it is essential to detect such changes in workload and temperature dynamics as early as possible, and adapt the ARMA model if necessary. We use sequential probability ratio test (SPRT) to detect the changes in the time series temperature data. SPRT provides the earliest possible detection of variations in time series signals [71].

The chapter is organized as follows. In Section 5.1 we provide the details of the ARMA predictor, and the online adaptation framework. We compare ARMA with other predictors in Section 5.3. Section 5.4 demonstrates how to predict workload dynamics for MPSoCs without thermal sensors. In Section 5.5, we explain all the thermal management techniques studied in this chapter, including *Proactive Temperature Balancing*. Section 5.6 provides the experimental methodology and results, and Section 5.7 summarizes the conclusions.

5.1 Temperature Prediction with Autoregressive Moving Averaging (ARMA)

In this section we provide an overview of our proactive temperature management approach, and explain the methodology for accurate temperature prediction at runtime. Proactive management adjusts the workload distribution on an MPSoC using the thermal forecast instead of reacting to thermal emergencies. In this way we reduce the performance cost associated with stalling or slowing down the cores after reaching a threshold.

Figure 5.1 provides an overview of our technique. We predict temperature t_n

steps into the future based on the temperature history observed through thermal sensors using an autoregressive moving average (ARMA) model. The scheduler then allocates the threads to cores to balance the temperature distribution across the die based on the predicted temperatures. The ARMA model utilized for temperature forecasting is derived based on a temperature trace representative of the thermal characteristics of the current workload. During execution, the workload dynamics might change and the ARMA model may no longer be able to predict accurately. To provide runtime adaptation, we monitor the workload through temperature measurements, validate the ARMA model and update the model if needed. The online adaptation method is explained in Section 5.2.

ARMA model assumes the modeled process is a stationary stochastic process, and that there is serial correlation in the data. In a stationary process the probability distribution does not change over time, and the mean and variance of the signal are stable. As workload characteristics are correlated during short time windows, and that temperature changes slowly due to thermal time constants, we assume the underlying data for the ARMA model is stationary. We adapt when the ARMA model no longer fits the workload. Thus, the stationary assumption does not introduce inaccuracy.

$$y_t + \sum_{i=1}^{p}(a_i\ y_{t-i}) = e_t + \sum_{i=1}^{q}(c_i\ e_{t-i}) \tag{5.1}$$

An ARMA(p,q) model is described by Equation 5.1. In the equation, y_t is the value of the series at time t (i.e., temperature at time t), a_i is the lag-i autoregressive coefficient, c_i is the moving average coefficient and e_t is called the noise, error or the residual. The residuals should be random in time (i.e., not autocorrelated), and normally distributed. p and q represent the orders of the autoregressive (AR) and the moving average (MA) parts of the model, respectively.

ARMA modeling has two steps: 1)*Identification and Estimation*, which consists of specifying the order and computing the coefficients of the model (coefficients are computed by software with little user interaction); 2) *Checking the Model*, where it is ensured that the residuals of the model are random and the estimated parameters are statistically significant.

Identification and Estimation:

During identification, we use an automated trial-and-error strategy. We start

by fitting the training data with the simplest model, i.e., ARMA(1,0), measure the "goodness-of-fit", and increase the order of the model if the desired fit is not achieved. At each iteration, to fit the data with the current order of ARMA model, coefficients are computed using a least-squares fit. Other methods can be utilized for coefficient estimation.

We use *Final Prediction Error (FPE)* [46] to evaluate the goodness-of-fit of the models. Once the FPE is below a predetermined threshold, we halt the trial-and-error loop. FPE is a function of the residuals and the number of estimated parameters. As FPE takes the number of estimated parameters into account, it compensates for the artificial improvement in fit that could come from increasing the order of the model. The FPE is given in Equation 5.2, where V is the variance of model residuals, N is the length of the time series, and $n = p + q$ is the number of estimated parameters in the model.

$$FPE = \frac{1 + n/N}{1 - n/N} \cdot V \tag{5.2}$$

Checking the Model:

For checking that the model residuals are random, or uncorrelated in time, we look at the *autocorrelation function (ACF)*. Autocorrelation is the cross-correlation of a signal with itself as a function of lag time, and is useful for finding repeating patterns in a signal if there are any. If model residuals are random, the ACF of all residuals (except for lag zero) should fluctuate close to zero. The residuals are assumed as random if the ACF for the majority of the trace is in between the pre-determined confidence intervals.

As an example, we apply the ARMA prediction methodology to a sample temperature trace. The trace is obtained through HotSpot [59] for a web server workload running on a system with a thermal management policy that swaps workload among hot and cold cores periodically, causing thermal cycles. We show a part of the trace in Figure 5.2, while the total length of the example trace is 200 samples long, sampled at every 100 ms. Using the first 150 samples of the data as the training set and $FPE << 1$, we form an ARMA(5,0) model. It should be noted that, for most of the real-life workloads we experiment with, much shorter training sets (i.e., 20-50 samples) are sufficient for forming an ARMA model with the desired fit.

We save the last 50 samples of the data to test our prediction method. We use the ARMA model to predict 5 steps (i.e., 500 ms) into the future. Figure 5.2 shows that

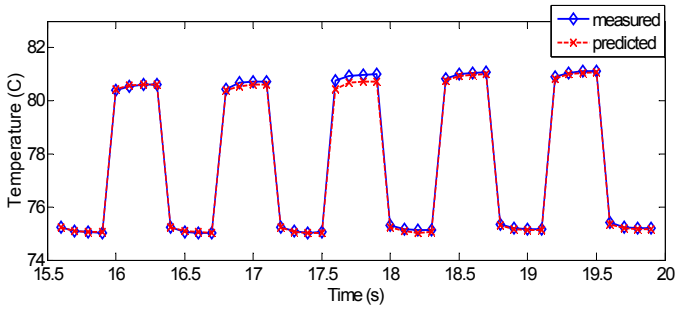

Figure 5.2: Temperature Prediction

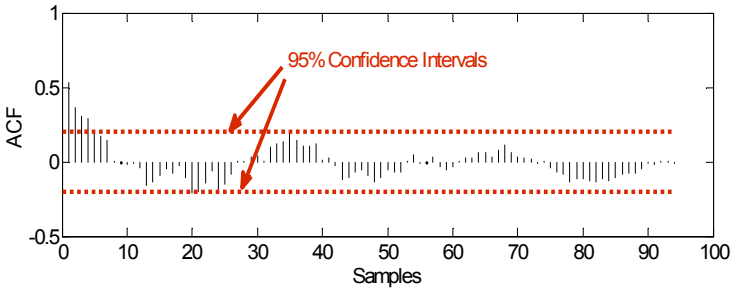

Figure 5.3: Autocorrelation Function of the Residuals

the prediction matches the observed values closely. For temperature curves with less temporal variation, designing an accurate ARMA predictor is even easier.

Figure 5.3 shows the ACF of the residuals for the model in Figure 5.2. Each sample refers to a 100ms interval. The horizontal lines in the figure show the 95% confidence intervals. In our automated methodology, we observe percentage of ACF values within the 95% confidence interval. If most of the ACF values fall within the 95% range, we declare that the residuals are random.

Computing the ARMA model has relatively low overhead. For example, in Matlab, an ARMA(p,0) model with $p \leq 10$ (no moving-average component) for a training data set of 50 samples is computed in less than 150ms, and an ARMA(p,q) model up to 5^{th} order is computed in less than 300ms. The computation and the validation of the model together takes between 250 and 500ms. Implementing the ARMA process in C/C++ and optimizing the source code is expected to reduce the overhead significantly.

Note that even though each core's ARMA model considers solely the temperature behavior of that particular core, the temperature trace of a core inherently takes into account the thermal behavior of the neighbor cores. This is due to the fact that temperature is not only dependent on the power consumption of a unit, but also on the floorplan and the power/thermal characteristics of other units on the die. While

it is possible to develop prediction techniques that consider a joint power/temperature profile of a set of units for forecasting, in our experiments we observe that considering each core's thermal trace individually result in accurate predictions.

5.2 Runtime Adaptation

ARMA models are accurate predictors when the time series data are stationary. Since the workload dynamics vary at runtime, the temperature characteristics may diverge from the training data used for forming the initial ARMA model. In order to adapt to changes in the workload, we propose monitoring the temperature dynamics and validating the ARMA model. When we determine the current workload deviates from the initial assumptions used for forming the ARMA model, we update the model on the fly.

We use Sequential Probability Ratio Test (SPRT) to detect changes over time in statistical characteristics of the residual signals. SPRT test on the residuals provides the earliest possible indication of anomalies [71], where as the anomaly in this case is defined as the residuals drifting from their expected distribution. Instead of using a simple threshold value for detection (e.g., setting a threshold for the standard deviation of the prediction error), SPRT performs statistical hypothesis tests on the mean and variance of the residuals. These tests are conducted on the basis of user specified false-alarm and missed-alarm probabilities of the detection process, allowing the user to control the likelihood of the missed detection of residual drifts or false alarms.

To perform online validation, we maintain a history window of temperature on each core. The window length is empirically selected based on thermal time constants and workload characteristics. To monitor the prediction capabilities of the model at runtime, for each new data sample we compute the residual by differencing the predicted data from the observed data. Our goal at runtime is to detect if there is a *drift* in residuals, where a drift refers to the mean of residuals moving away from zero (Recall that for an ARMA model with good prediction capabilities, the residuals should fluctuate close to zero). Detecting the drift quickly is important for maintaining the accuracy of the predictor, as such a drift shows that the model no longer fits the current temperature dynamics.

Specifically, we declare a drift when the sequence of observed residuals appears to be distributed about mean $+M$ or $-M$ instead of around 0, where M is our preassigned

system disturbance magnitude. A typical value for M is $(3 * \sqrt{V})$ [28], where V is the variance of the residuals in the training data set.

$$R(t) = T_i(t) - T_i^{'}(t) \tag{5.3}$$

At time instant t, the residuals (R) can be computed by Equation 5.3, where $T_i^{'}(t)$ is the prediction and $T_i(t)$ is the measurement. SPRT then decides between the following two hypotheses:

1. H_1: $R(t)$ is drawn from a probability density function (pdf) with mean M and variance σ^2.
2. H_2: $R(t)$ is from a pdf with mean 0 and variance σ^2.

In other words, we detect that there is a drift if SPRT decides on H_1. If H_1 or H_2 is true, we wish to decide on the correct hypothesis with probability $(1 - \beta)$ or $(1 - \alpha)$, respectively, where α and β are false alarm and missed alarm probabilities. Small values such as 0.01 or 0.001 are used for α and β.

$$LR_N = ln\frac{p[R(1), R(2), ..., R(N)/H_1]}{p[R(1), R(2), ..., R(N)/H_2]} \tag{5.4}$$

SPRT is applied to detect the drift (i.e., anomaly) in residuals by computing the log likelihood ratio in Equation 5.4, where $p(./H_2)$ is the joint density function assuming no fault, and $p(./H_1)$ is the joint density function assuming fault, and N is the number of observations. If $LR_N \geq B$ we accept H_1, meaning that the residuals show significant change from the assumptions; and if $LR_N \leq A$ we accept H_2. If one of the hypotheses is accepted, the SPRT computation is restarted from the current sample. Otherwise (i.e., $A < LR_N < B$) we continue the measurements. The bounds A and B are defined as in Equation 5.5.

$$A = ln(\frac{\beta}{1-\alpha}) \qquad B = ln(\frac{1-\beta}{\alpha}) \tag{5.5}$$

Following the derivation provided in [28], the value of SPRT can be represented as shown in Equation 5.6, where M is the system disturbance magnitude as defined previously, and σ^2 is the variance of the residuals in the training set.

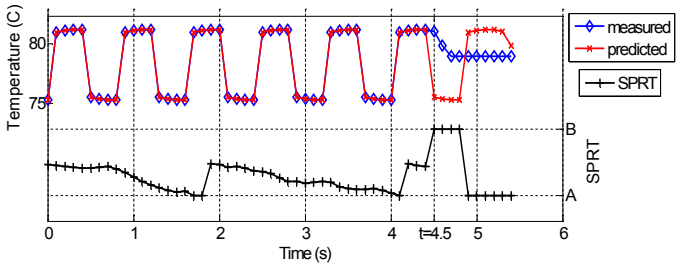

Figure 5.4: Online Detection of Variations in Thermal Characteristics

$$SPRT = \frac{M}{\sigma^2} \sum_{i=1}^{N} (R(i) - \frac{M}{2}) \tag{5.6}$$

Note that M and σ^2 values are computed at the beginning, and then fixed until the ARMA model is updated. At runtime, during each sampling interval, the SPRT equation effectively performs one addition and one multiplication. Because of the simplicity of computation shown in Equation 5.6, the cost of computing SPRT after each observation is very low (negligible in our measurements). Moreover, as shown in [71], there is no other procedure that has the same error probabilities with shorter average sampling time than SPRT. We have picked SPRT as the online monitoring tool in this work due to both its guarantee for fast detection of changes and low computation overhead.

In Figure 5.4, we demonstrate a case where the temperature dynamics change, and the SPRT detects this change immediately (see $t = 4.5s$ in the figure). A and B correspond to the SPRT thresholds of ± 6.9068 for α and β values of 0.001. When $SPRT >= 6.9068$ (i.e., $LR_N \geq B$), we declare that the residuals have a drift from the training data, initiating the computation of a *new* ARMA model. Recall that when $SPRT <= -6.9068$ (i.e., $LR_N \leq A$) we accept the hypothesis that the mean of the residuals is 0. In both cases, the SPRT computation is restarted.

We also compared SPRT detection with monitoring the standard deviation of the residuals. The prediction capability of an ARMA model can be examined by computing the standard deviation of the prediction error (i.e., difference between actual measurements and predictions). If the dynamic characteristics of the temperature time series can be well represented by the model, the standard deviation of the associated prediction error should be relatively small. It is generally recommended to keep the standard

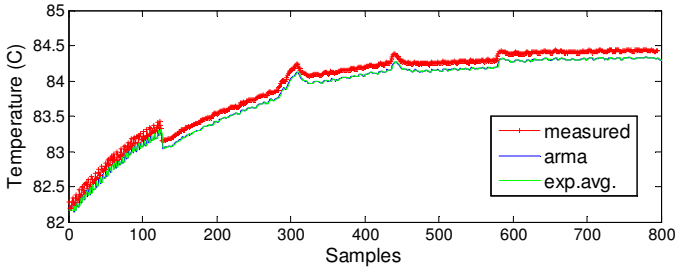

Figure 5.5: Comparison of Predictors - Stable Temperature

deviation of prediction errors less than 10% of the standard deviation of the original signal. This condition implies that the ARMA model is able to capture more than 90% of the underlying dynamics of the system. Using a 10% threshold, the standard deviation method can quickly detect the change in temperature dynamics in the case of abrupt changes such as in Figure 5.4. However, for gradual shift in thermal dynamics, it may fail to capture the drift immediately. SPRT guarantees the fastest detection for the given false and missed alarm probabilities.

5.3 Comparison with Other Predictors

In this section, we compare ARMA with various predictors in terms of their prediction and adaptation capabilities, and their computation and hardware overhead.

5.3.1 Exponential Averaging

A well-known method for prediction is exponential moving averaging. In Figures 5.5 and 5.6, we compare ARMA prediction with exponential average prediction for an execution slice of a highly utilized web server workload and the previous trace used in Figure 5.2, respectively. The exponential average predictor estimates the current value of the series as: $y_t = \alpha T_{t-1} + (1-\alpha)y_{t-1}$, where y_t is the predicted temperature (exponential average) at time t, T_{t-1} is the measured temperature at time t-1, and α is a constant ($0 \leq \alpha \leq 1$). We take $\alpha = 0.9$ in Figure 5.5 and $\alpha = \{0.5, 0.9\}$ in Figure 5.6.

When we have relatively stable temperature, exponential average predictor works well, providing almost the same values as the ARMA predictor in Figure 5.5. However, when there are rapid temperature changes, such as thermal cycling, exponential average predictor performs poorly, such as in Figure 5.6. In addition, even though exponential average predictors with $\alpha = 0.9$ and $\alpha = 0.5$ perform very similarly in the first example,

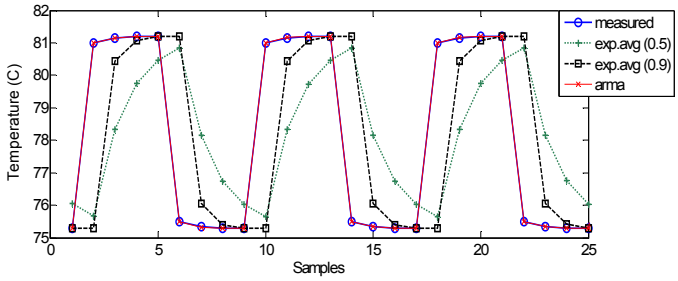

Figure 5.6: Comparison of Predictors - Thermal Cycling

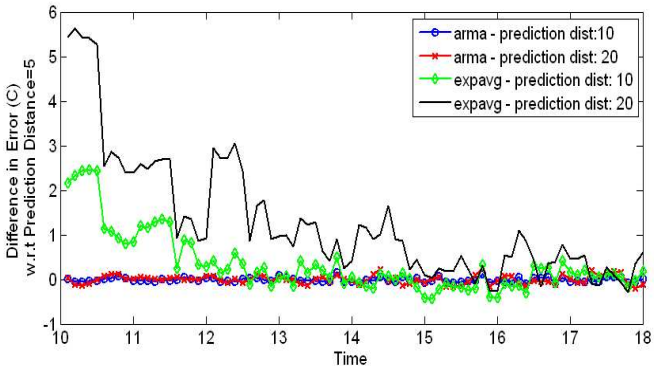

Figure 5.7: Predicting Further Ahead with Exponential Averaging

there is significant effect of the α value in the thermal cycling case, which would require the user to determine α accordingly. Contrarily, ARMA predictor has an automated process of forming the model with high accuracy. The overhead of evaluating the ARMA or the exponential average model at runtime is very similar, as both models only compute a polynomial equation for each sample.

The ideal number of steps to predict ahead depends on the system and workload characteristics. In our experiments, we predict 500 ms (i.e., 5 steps) into the future, as this prediction distance provides good results for our proactive thermal management policies. However, for different system and workload characteristics, the preferred prediction distance may vary. For example, for systems with less variant workload, predicting further ahead and using a lower sampling rate for polling the temperature sensors would be sufficient.

For the experiment in Figure 5.7, we increase the prediction distance to 10 and 20 steps to evaluate the accuracy of ARMA and exponential averaging predictors as a function of the prediction distance. When the goal is forecasting several time steps into the future, the prediction accuracy of exponential averaging degrades significantly. For

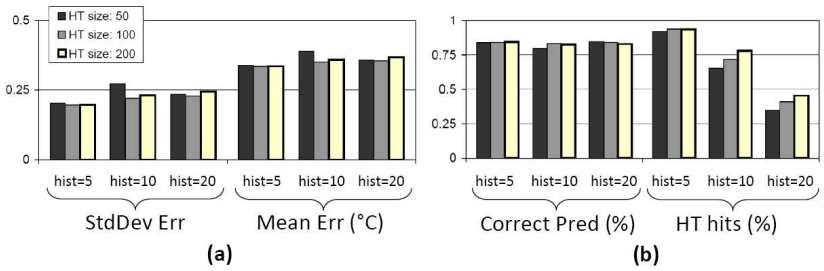

Figure 5.8: Accuracy-Size Trade-Off for the History Predictor

this experiment, we use a 200 sample temperature trace for a CPU bound SPEC 2000 suite workload, where the temperature is changing within a 1.5^oC range. The plot shows the difference of error (in ^oC) in comparison to predicting 5 steps into the future with the same predictor. While ARMA predictor's accuracy is stable, the error margin of the exponential predictor increases considerably when predicting ahead.

5.3.2 History Predictor

In Section 5.1, we show that it is possible to predict future temperature accurately based on the previous thermal measurements. Following this insight, we built a history predictor. A history predictor is similar to a global branch predictor, and it consists of a shift register that tracks the last few observed values. The length of the history is specified by shift register depth. At each sampling period, the register is updated with the last measurement. This updated shift register content is used to index a history table (HT). The HT holds several previously observed thermal patterns, with their corresponding next value predictions based on previous experience. Shift register index is associatively compared to the stored valid HT tags, and if a match is found, the corresponding HT prediction is used as the final prediction. An invalid entry is kept for each tag to track the ages of different HT tags for a least recently used (LRU) replacement policy when the HT is full. A -1 entry denotes the corresponding tag contents and prediction are not valid. This predictor is similar to the Global History Predictor used for predicting power phases in [32]. When the shift register does not hit the history, the predictor behaves like a last-value predictor, and assumes the future temperature value will be the same as the last observed temperature.

One issue with the history predictor is the precision of temperature data. We also perform experiments where we store temperature readings with one or two decimal places. However, even for relatively stable temperature profiles, obtaining a reasonable

percentage of hits on the HT has not been possible when we consider the decimal places. In addition, even when we maintain one decimal digit, the required HT size for predicting with high accuracy becomes considerably large (i.e., we would have to have new entries in the table to accommodate even slight changes in the decimal digit). For this reason, for the history predictor we round the temperature measurements to nearest integer values, and only predict temperature in integers.

In Figures 5.8 (a) and (b), we show the accuracy for various history table (HT) sizes and history lengths. In this experiment, we use the same temperature trace in Figure 5.7, and predict 5 steps ahead. In (a), we compare the standard deviation of error and mean error (in ^oC) for the prediction, where error is the difference of measured trace and predictions. For this workload, we observe that increasing the history length does not bring much benefit; however, increasing the table size reduces the magnitude of errors. In (b), we demonstrate the correct prediction ratio (with respect to the integer temperature trace) and the hit rate for the history table. While increasing the history length reduces the hit rate as expected, the accuracy does not get affected by this. This is due to the fact that for stable profiles, last value predictor compensates well when the history predictor cannot predict. Note that, increasing the table size over 100 does not bring additional benefits, which motivates using a small-size table to achieve enough accuracy with lower hardware overhead.

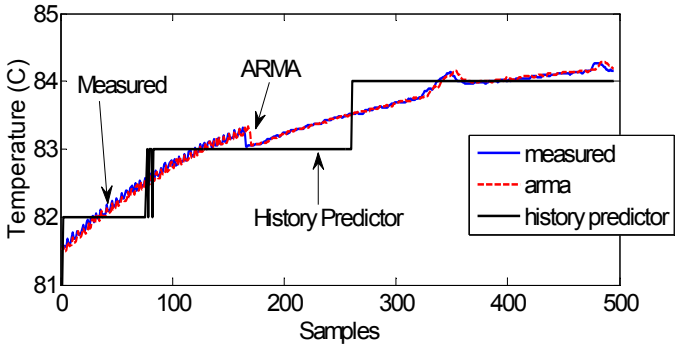

Figure 5.9: Comparison of ARMA and History Predictor

Figure 5.9 compares the ARMA predictor and the history predictor (HP with HT size of 100 and history length of 5) for predicting 5 steps ahead (i.e., 500ms). We observe that the ARMA captures the thermal dynamics almost exactly, while the history predictor can predict the integer value of the temperature with reasonable accuracy. For repeating patterns of workload, such as several applications being time-multiplexed on

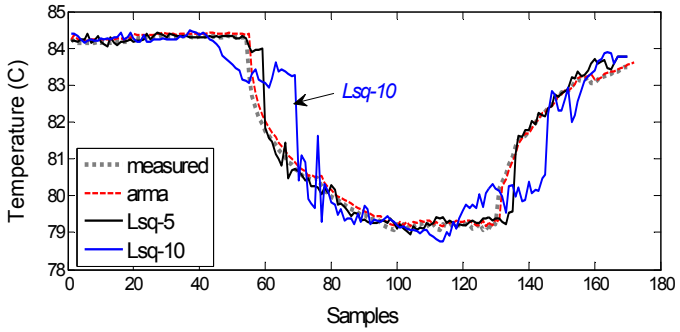

Figure 5.10: Comparison of ARMA and Recursive Least Squares Predictor

a core, and stable thermal profiles, the history predictor can predict with high accuracy and does not require a training phase (except for the first time an application is run), provided that the history table is large enough to maintain the entries associated with all of the applications.

5.3.3 Recursive Least Squares

Another recently proposed temperature prediction method is using recursive least squares [73]. In Figure 5.10, we compare the prediction accuracy of the least squares (Lsq) approach with ARMA. We train both predictors with 50 samples of the temperature data. While Lsq with the prediction distance of 5 (shown as *Lsq-5*) has similar accuracy as ARMA (prediction distance=10), the accuracy of Lsq drops rapidly when we increase the forecasting distance (t_n) to 10 steps. This trend continues even more dramatically for higher t_n.

Note that both of the above methods can predict the data in the history window they are trained with the desired degree of accuracy. If one keeps adding enough terms, it is even possible to fit through every single observation in the history window. Typically we would not want to do that; however, because usually there is random measurement noise on the time series, and there is no value to learning the noise. Thus, the differentiating point of least squares and ARMA arises when we are forecasting further into the future. As we increase t_n, least squares does significantly worse than ARMA. The reason is that as soon as we predict more than a few time steps into the future, the term with the biggest exponent in the least squares fitting function dominates and the prediction accuracy degrades from that point on.

Another important advantage of ARMA in comparison to least squares is in overhead. Recursive least squares method continuously updates the coefficients of the model

as new data arrives (otherwise accuracy drops), whereas the SPRT support enables us to update the model only when it is necessary. In addition, the length of the polynomial in the least squares estimation needs to be set manually, which can unnecessarily increase computation overhead if set to a larger value than needed. On the other hand, we use an automated and fast trial-and-error strategy for automatically setting the number of terms in the ARMA model.

5.4 Workload Prediction

The predictors discussed above utilize an on-chip telemetry infrastructure, which provides temperature measurements at the desired granularity. In many systems we may not have a thermal sensor for each core, or sensors may degrade and fail during the system lifetime. In this section we discuss how to predict workload parameters for applying a proactive management strategy for such cases.

We demonstrate prediction of two parameters: 1) Core utilization, 2) IPC of committed instructions. Core utilization is a good measure of how busy the core is and hence provides an insight for the power consumption, especially in multi-threaded systems, where we may not have access to measuring per-thread IPC. For single threaded systems, IPC tends to have a strong correlation with the power consumption. While such performance metrics may not directly reflect the thermal behavior of cores, they still provide an estimation of whether the power consumption is increasing or decreasing in the near future. Therefore, forecast of future workload can be utilized to perform proactive temperature management, assuming a correlation between high utilization/IPC and high power consumption.

Figures 5.11 and 5.12 show traces of core utilization and committed IPC, respectively, and the prediction results obtained by ARMA. The core utilization results are collected for medium utilized web application on a multi-threaded system. The IPC trace belongs to `bzip` running on a single-threaded architecture. Both predictors are trained using 150 samples, and prediction is performed for the following 50 samples. Note that the workload parameters may have short term spikes due to changing application characteristics, while these do not typically get reflected in the temperature response due to the thermal time constants. This is especially the case for core utilization. To achieve more accurate prediction for core utilization, we applied a smoothing function (i.e., moving averaging) to the workload traces. The smoothed-out utilization

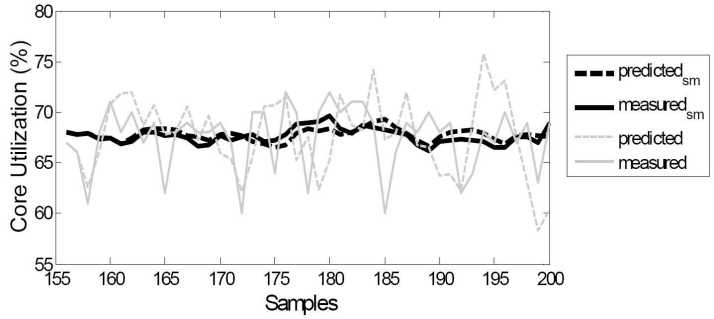

Figure 5.11: Prediction of Core Utilization

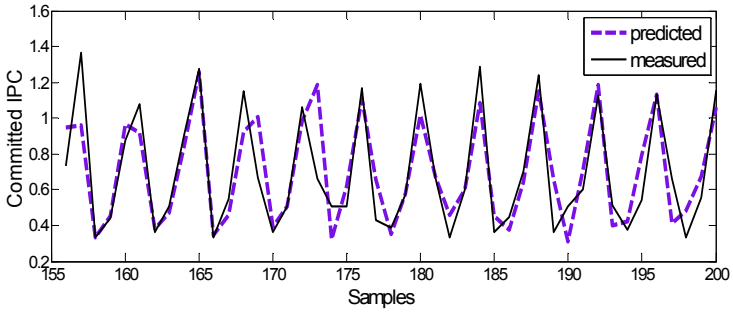

Figure 5.12: Prediction of Committed IPC

and prediction signals are demonstrated with the subscripts *sm* in Figure 5.11. For the original trace, the accuracy of utilization prediction is significantly lower than temperature prediction. The predictor is more accurate when the data is smoothed-out first.

Even though bzip is a highly IPC-variant benchmark, Figure 5.12 shows that IPC can be predicted with high accuracy. Note that applications typically have different phases of performance, and SPRT would detect such a change immediately. The substantial accuracy difference between predicting IPC and core utilization is mainly due to the difference between observing a single-thread and observing multiple threads at the same time. The core utilization results are collected on a multi-threaded system, where the core is running a set of threads rather than a single application.

5.5 Proactive Job Allocation

This section discusses the details of all the management policies we implement for evaluation, including our novel *Proactive Temperature Balancing* technique. We consider both single-threaded and multi-threaded systems while studying thermal management

policies. In the system model for the multi-threaded systems, each core is associated with a *dispatching queue*, which holds the threads allocated to that core. This is the typical abstraction used in modern multicore OS schedulers, which are based on multilevel queuing. The dispatcher allocates the incoming threads to queues based on the current policy.

Similar to previous chapters, the default policy we evaluate (default policy in modern OSes such as Solaris) is `Dynamic Load Balancing (DLB)`. DLB assigns an incoming thread to the core it ran previously, if the thread ran recently. If the thread has not run recently, then the dispatcher assigns it to the core that has the lowest priority thread in the queue. The dispatcher first tries to assign the thread based on locality (e.g., if several cores are sharing a cache or on the same chip) if possible. If there is significant imbalance among the queues at runtime, the threads are migrated to have more balanced utilization.

Power Management:

Many current MPSoCs have power management capabilities to reduce the energy consumption. Even though the power management techniques do not directly address temperature, they affect the thermal behavior significantly. We implement two commonly used power management methods; `Dynamic Power Management (DPM)` and `Dynamic Voltage-Frequency Scaling (DVS)`. For DPM, we utilize a fixed timeout policy, which puts a core to sleep state if it has been idle longer than the timeout period. We set the timeout as the breakeven time [37]. The DVS policy observes the core utilization over a given length of recent history, and reduces the voltage/frequency proportionally (using the discrete settings available).

Reactive Thermal Management:

Several reactive thermal management techniques have been proposed in the literature (e.g., [26]). In this work we implement some of the most commonly used methods.

Reactive Thread Migration (R-Mig) migrates the workload from a core if the threshold temperature is exceeded to the coolest core available. In single threaded systems, this correspond to migrating the currently running job or swapping the jobs among the hot and cool cores. In multi-threaded environment, the technique migrates the current threads in the hot core's dispatch queue to other cool cores, or swaps threads among hot and cool cores.

Reactive DVS (R-DVS) reduces the voltage/ frequency (V/f) setting on a core if

the threshold temperature is exceeded, similar to the frequency scaling approach in [59]. We assume three built-in V/f states in our experiments. The policy continues to reduce the (V/f) level at every tick as long as the temperature is above the threshold. When the temperature is below the critical threshold, then the V/f setting is increased.

Proactive Thermal Management:

The proactive methods utilize the temperature prediction introduced in Section 5.1. The motivation behind proactive management is to avoid thermal emergencies before they occur, and thus to minimize the adverse effects of hot spots and temperature variations at lower performance cost.

In the workload allocation techniques we propose, we do not change the priority assignment of the threads or the time slices allocated for each priority level. Our work focuses on finding effective dispatching methods to reduce temperature induced problems without affecting performance.

Proactive Thread Migration (P-Mig) moves workload from cores that are projected to be hot in the near future to cool cores. `Proactive DVS` reduces the V/f setting on a core if the temperature is expected to exceed the critical threshold. These two policies are the same as their reactive counterparts, except that they get triggered by the temperature estimates instead of the current temperature.

Proactive Temperature Balancing (PTB) follows the principle of locality (i.e., allocating the threads on the same core they ran before) during initial assignment as in the default policy. At every scheduler tick, if the temperatures of cores are predicted to have imbalance in the next interval, threads waiting on the queues of potentially hotter cores are moved to cooler cores. This way, the thermal hot spots can be avoided, and the gradients are prevented by thermal balancing.

In a single-threaded system, we bound the number of migrations to avoid the unnecessary performance cost. Migration of the jobs on all the hot cores can cause thermal oscillations. We start performing migrations from the hottest core, and migrate only if the workload on the hot core's neighbors have not been migrated during the current tick. Note that in a multi-threaded environment, threads waiting in the queue are moved unless the threshold is already exceeded, so migration does not stall the running thread. This is in contrast to moving the actively running threads in thread migration policies discussed above. As the default load balancing policy already moves the waiting threads if there is an imbalance among the queues, our technique does not

introduce additional overhead. In *Proactive Temperature Balancing* for multi-threaded systems, the number of threads to migrate is proportional to the spatial temperature difference among the hot core and the cool core.

5.6 Results

This section evaluates the thermal management techniques discussed in the previous section. In the results, DLB is the default load balancing policy, R-Mig (P-Mig) and R-DVS (P-DVS) refer to the reactive (proactive) migration and voltage scaling, respectively, and PTB is our proactive temperature balancing policy (i.e., combined with ARMA predictor). All predictors in this section predict 5 steps ahead (i.e., 500ms assuming a 100ms sampling rate).

We show two sets of experimental results. The first set is based on the UltraSPARC T1 processor [45]. In the second set of results, we use the phase-based architecture-level simulation framework to simulate performance, power and temperature, and provide results on a hypothetical high-performance 16-core architecture manufactured at 65nm.

The threshold temperature for the management policies is 85^oC, which is considered a high temperature for our system. In this section, the hot spot results demonstrate the percentage of "time spent above 85^oC". The spatial gradient results summarize the percentage of time that gradients above 15^oC occur. We report the temporal fluctuations of magnitude above 20^oC. ΔT values we report are computed over a sliding temperature history window (i.e., maximum ΔT in the history window) and averaged over all cores.

5.6.1 UltraSPARC T1 Implementation

The first set of experimental results are based on the UltraSPARC T1 [45]. The experimental flow consists of gathering workload traces, applying policies (scheduling, DVS, etc.) on the given workload, computing the corresponding power traces, and finally calculating the temperature response, as discussed in Section 3. We utilize the same real-life workload demonstrated in Table 3.5.

The results marked as *real implementation* refer to our implementation of the policies in Solaris, where we run the workload on the UltraSPARC T1 in real-time. Some of our policies utilize temperature readings from all cores, and UltraSPARC T1 does not contain an individual sensor for each core. To obtain detailed thermal data

in synchronization with the scheduling experiments, using a shared file we pipe the utilization data collected from the target machine to another computer that is running the thermal simulator. The utilization data is converted into the equivalent power trace by the thermal simulator, the temperature results are computed for the next interval, and then are passed back to the target system (which is running the thermal management policies). A separate computer in the private network is assigned to run the thermal simulator to avoid interfering with the workload dynamics on the target system. In the *real implementation*, the core utilization statistics are passed to the thermal simulator at every 1 second interval, and the thermal simulations are sampled at every 100 ms. To implement thread migration, we utilize the existing migration routine in the OS dispatcher, and include additional temperature-induced triggers accordingly.

The results marked as *simulator* are from our simulation infrastructure attached to the power/thermal model, where we use the real-life workload traces again, but this time implement the scheduling policies within the simulator that is a replica of the multicore system model. Again, the temperature sampling rate is set at 100 ms.

First we provide the *simulator* results. Table 5.1 shows a detailed analysis of the hot spots observed on the system for each workload, and also the average performance results. We show the percentage of time spent above 85^oC for all the workloads, and the average results for the cases with no power management (No PM) and DPM. The performance results shown in the table are normalized with respect to the default policy's performance. We compute performance based on the average delay we observe in the thread completion times. Reactive migration of workload or applying temperature triggered DVS cannot eliminate all the hot spots, especially for workloads with medium to high utilization level. Performing migration or DVS proactively achieves significantly better results, while also reducing the performance cost. The cost is lower with the proactive approaches as they maintain the temperature at lower levels, requiring fewer overall number of migrations or shorter periods of DVS. Note that, once a temperature threshold is reached, execution at the default speed is not allowed on a core until the temperature is lowered. Also, when system is highly utilized, swapping threads may not reduce the temperature sufficiently, and frequent threshold triggers may occur as new threads arrive.

Our technique, PTB, achieves very similar results to proactive DVS while it has much better performance. DPM reduces the thermal hot spots to some extent

Table 5.1: Thermal Hot Spots and Performance (*Simulator*)

	no PM						DPM					
Workload	DLB	R-Mig	P-Mig	R-DVS	P-DVS	**PTB**	DLB	R-Mig	P-Mig	R-DVS	P-DVS	**PTB**
Web-med	25.9	12.9	5.9	7.7	3.3	3.8	19.5	10.9	3.4	4.6	2.0	2.5
Web-high	39.1	22.1	13.3	19.2	10.4	10.6	37.4	21.6	10.7	14.8	8.4	8.5
Database	8.3	2.1	1.2	1.5	1.1	1.0	4.6	1.5	0.0	1.1	0.0	0.0
Web&DB	32.4	15.3	7.1	10.7	5.2	4.8	27.8	13.2	7.7	6.7	4.8	4.6
gcc	7.2	1.8	1.5	0.5	1.3	0.7	3.8	1.3	0.0	0.1	0.0	0.0
gzip	2.9	0.6	0.0	0.1	0.0	0.0	1.3	0.5	0.0	0.0	0.0	0.0
Mplayer	4.9	0.7	0.0	0.4	0.0	0.0	1.7	0.5	0.0	0.1	0.0	0.0
Mplayer&Web	13.3	9.4	5.3	4.9	2.4	2.1	8.9	7.2	5.2	4.1	1.2	1.1
AVG	16.8	8.1	4.3	5.6	3.0	2.9	13.1	7.1	3.4	3.9	2.1	2.1
AVG Perf.	1.00	0.96	0.97	0.89	0.91	0.98	1.00	0.95	0.96	0.87	0.90	0.97

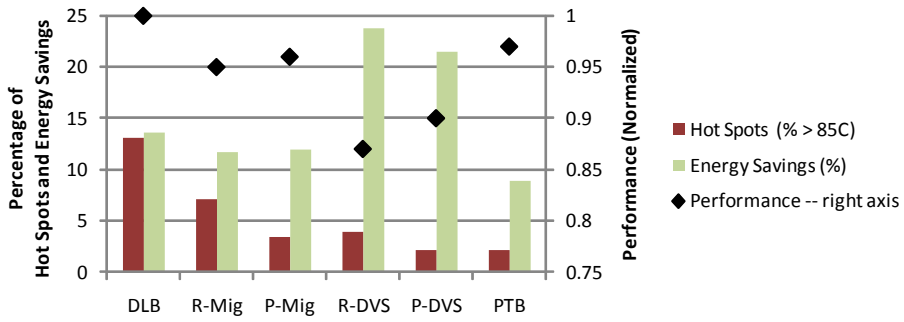

Figure 5.13: Energy Savings, Hot Spots, and Performance - with DPM (*Simulator*)

as it reduces the temperature when the system has idle time. Performing proactive temperature management results in the best thermal profile among the techniques when there is DPM; i.e., 83% reduction in hot spots in comparison to DLB.

We next look at how the energy savings obtained with DPM change depending on the policy. Figure 5.13 shows the energy savings with respect to DLB without power management. The plot also includes the percentage of hot spots and the performance of each policy (hot spots and performance values are reported in Table 5.1 as well). Among the workload allocation/migration policies, DLB has the highest savings in energy. R-Mig, P-Mig, and PTB balance the workload more than DLB does to reduce thermal problems. On the other hand, as DLB clusters the workload more than the migration policies, it achieves longer continuous idle time slots and improves energy savings with DPM. DLB achieves 13.6% savings on average across all the workloads, while P-Mig and R-Mig reduce energy consumption by close to 12%. PTB performs dramatically better than the other migration-based policies in terms of reducing the thermal problems, while still obtaining 8.9% savings when combined with DPM. DVS policies considerably increase the savings; e.g., 21.5% for P-DVS and 23.7% for R-DVS, when combined with DPM. However, recall that DVS significantly increases the execution time. Thus, while DVS reduces the energy consumption of the cores, due to prolonged activity of memories and other components, the total energy consumption of the system may not benefit as much from DVS. In addition, a significant portion of the total energy costs in current servers is due to cooling costs, which we do not consider in this computation.

Figure 5.14 shows the average percentage of time we observe thermal cycles above 20^oC. We also plot the workload with the maximum thermal cycling, Web-med, for comparison. We only consider the case with DPM for the thermal cycling results, as

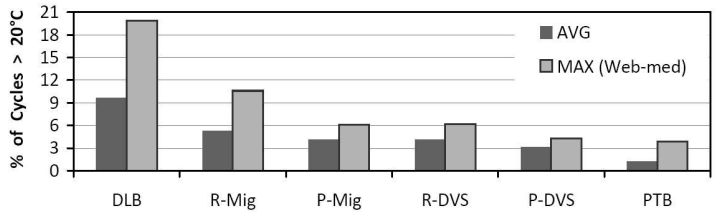

Figure 5.14: Temperature Cycles - with DPM (*Simulator*)

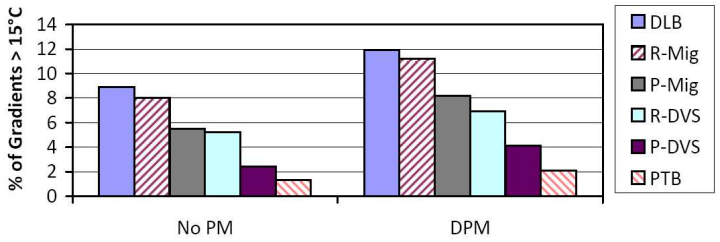

Figure 5.15: Spatial Gradients (*Simulator*)

putting cores to sleep state creates larger cycles. Our technique achieves very significant reduction in thermal cycles, i.e., to around 1% in the average case, as it continuously balances the workload among the cores according to their expected temperature. As reactive techniques take action after reaching temperature thresholds, they cannot avoid the temperature imbalance in time as well as our technique. P-DVS and PTB perform very similarly; however it should be noted that the performance cost of PTB is less than DVS.

Figure 5.15 shows the average percentage of time large spatial gradients above 15^oC occurred while running the policies. DPM creates larger gradients due to the low temperatures on the cores that go into the sleep state. Proactive temperature balancing can almost eliminate large gradients by reducing their frequency to below 2% in average. Proactive DVS is the second best policy for reducing the on-die variations.

To show the effect of runtime adaptation on the accuracy of our technique, we run traces of different workloads sequentially and compute the temperature statistics. As examples, in Table 5.2, we show the results for running the following combinations of workload with the *PTB* policy: (A) Web-med followed by Web&DB, (B) Mplayer followed by Web-med. We show the percentage of hot spots, cycles and gradients for the individual workloads, and also for the combined workloads for the case with DPM. We run equal lengths of each benchmark in the combined workloads. We see that the

Table 5.2: Temperature Results for Combined Workloads (*Simulator*)

	Hot Spots (%)	Cycles(%)	Gradients(%)
Web-med	2.6	4.5	4.4
Web&DB	4.6	2.9	5.7
Mplayer	0	0.1	0.9
(A) Web-med, Web&DB	3.7	3.7	5.0
(B) MPlayer, Web-med	1.3	2.2	2.7

percentage of hot spots and variations of the combined workload are close to the average values of running the individual benchmarks. Thus, PTB can adapt to workload changes without negatively affecting the thermal profile.

We next discuss results collected on the *real implementation*, where we implement our technique in the Solaris task dispatcher running on an UltraSPARC T1 system. On the real implementation, we simulate DPM effects on temperature using HotSpot as with the simulator, and assume the transition overhead among active and sleep states has negligible overhead. As the system does not have DVS capabilities, we simulate the thermal behavior for the default policy (DLB), reactive and proactive migration, and our policy (PTB), running the benchmark set described previously.

In Table 5.3, we show the distribution of hot spots, comparing various benchmarks. The combination workloads (A) and (B) are described in Table 5.2 above. We observe that PTB can reduce the hot spots by 60% in average in comparison to reactive migration, and 20 to 30% with respect to proactive migration. Workloads with low utilization, such as Mplayer, do not have a significant percentage of high temperatures. However, for hotter benchmarks PTB achieves dramatic reduction in the occurrence of hot spots.

Table 5.3: Hot Spots (*Real Implementation*)

	no PM				DPM			
Workload	DLB	R-Mig	P-Mig	PTB	DLB	R-Mig	P-Mig	PTB
Web-med	25.7	14.3	6.2	4.8	19.5	12.4	4.8	3.9
Database	8.4	3.5	1.8	1.3	4.6	3.1	1.5	1.2
Web&DB	32.4	15.7	8.1	6.0	27.4	14.7	9.1	5.8
Mplayer	4.9	1.5	0.7	0.9	1.9	2.0	1.5	1.2
(A)	17.2	9.1	4.9	4.1	23.7	14.2	7.9	5.1
(B)	15.6	8.5	4.5	3.7	10.7	8.0	3.7	3.6
AVG	17.4	8.8	4.4	3.5	14.6	9.1	4.8	3.5

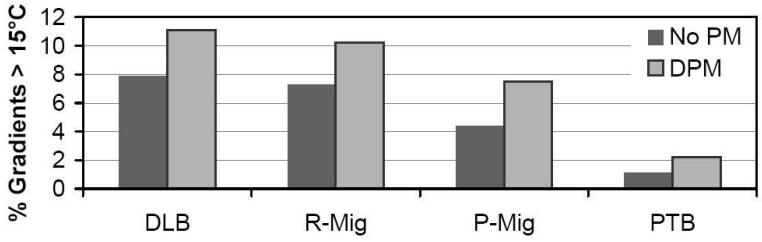

Figure 5.16: Spatial Gradients (*Real Implementation*)

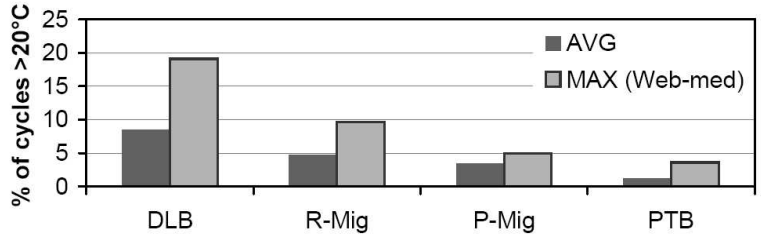

Figure 5.17: Thermal Cycles - with DPM (*Real Implementation*)

Figures 5.16 and 5.17 demonstrate the average frequency of spatial gradients and thermal cycles on our real system implementation. These results agree with the simulation results that, PTB reduces the thermal variations more effectively in comparison to other proactive and reactive techniques.

As the *real implementation* on UltraSPARC T1 runs multi-threaded workloads, we do not use an IPC-based performance metric. Evaluating the performance of multi-threaded workloads using IPC is prone to inaccuracy [2]. This inaccuracy is due to the assumption that instructions per program remains constant across all executions, whereas the instruction path of multi-threaded workloads running on multiple processors can vary substantially. Thus, to evaluate the performance of the various techniques we implement, we use the "Load Average" metric, as in Chapter 4. Recall that as load average grows, performance degrades.

Figure 5.18 demonstrates the performance values for the policies, normalized relative to the default policy (i.e., DLB). Proactive temperature balancing is able to achieve better thermal profiles than other policies with less performance cost. This is because PTB first attempts to migrate the threads waiting in the dispatch queue, as opposed to stalling and migrating actively running threads. For example, for the workload `Web-med`, in the default case, the number of migrations of *active threads* is 0.004

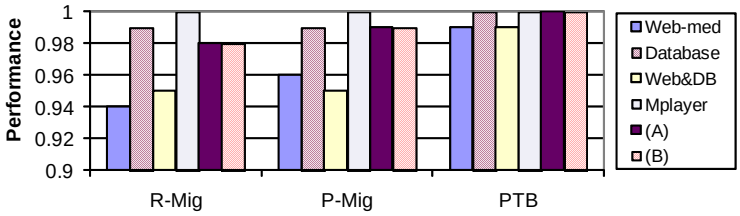

Figure 5.18: Normalized Performance (*Real Implementation*)

per 1000 instructions. Reactive migration (R-Mig) increases this number to 0.009/(1K instructions). Proactive migration causes fewer number of migrations than R-Mig (0.008/ 1K-instructions), as the temperature becomes more stable and the frequency of thermal emergencies decrease. PTB reduces this number further to 0.0046, which is only slightly higher than the default case. Note that, the number of migrations of threads that are waiting in the queue are higher with PTB; however the performance cost of such migrations are much lower.

Lastly, to show the effect of prediction accuracy on thermal behavior, we implement the proactive temperature balancing (PTB) using least squares prediction (LSQ) and history predictor (HISTORY), and compare the results against performing PTB with ARMA. Figure 5.19 compares the percentage of hot spots observed with all the predictors. For this experiment, we ran the following benchmarks sequentially in the given order: Web-medium, Web-high, Web&Database and Database. Each benchmark runs for an equal amount of time. PTB with ARMA achieves a better thermal profile than PTB with other predictors. This advantage is mainly a result of the longer adaptation period of the history predictor and least squares predictor when the workload changes. SPRT detects the change immediately and computes a new ARMA model, whereas the other predictors go through a training period before starting accurate predictions.

5.6.2 Phase-Based Architecture-Level Simulator

To study the effect of reactive/ proactive thermal management strategies in larger MPSoCs with higher performance cores, we use the phase-based architecture-level simulation framework discussed in Chapter 2 in addition to the results we collect on UltraSPARC T1. Following the trend of integrating an increasing number of cores on a single die, e.g., Sun's 16-core Rock processor [68] and Intel's Larrabee with up to 32 cores [56], the CPU we model is a homogeneous 16-core multiprocessor manufactured at

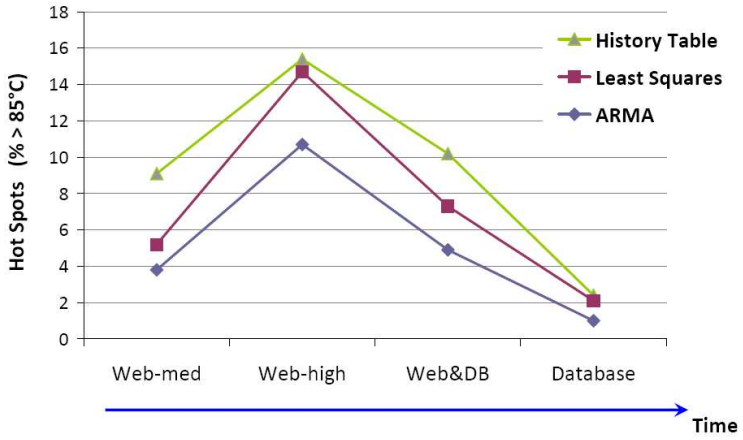

Figure 5.19: Proactive Balancing Results for Various Predictors

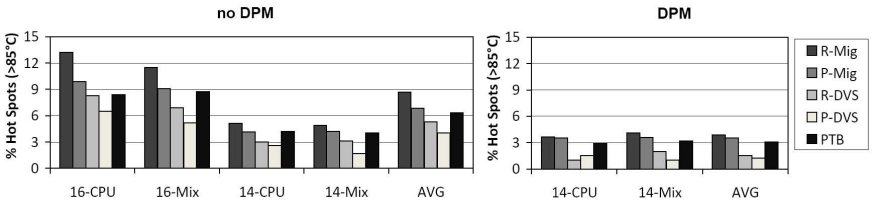

Figure 5.20: Thermal Hot Spots (16-Core System)

65nm. The floorplan for this CPU is provided in Figure 2.2.

We use a subset of the SPEC-based workloads outlined previously in Table 2.4 for this set of experiments. The four benchmarks with 16 and 14 threads we utilize in this section are shown in Table 5.4. These benchmarks have different intensity of CPU and memory instructions to create representative traces for a wide range of real-world applications.

Next, we provide results on how the policies affect the thermal behavior and performance of the 16-core architecture. Figure 5.20 demonstrates the frequency of hot spots for R-Mig, P-Mig, DVS and PTB. We use the single-threaded version of the policies for this part of the experiments.

Unlike the multi-threaded simulations, in Figure 5.20 we see that DVS can reduce the frequency of hot spots more effectively. However, this comes at a performance cost, which we investigate later. On our 16-core architecture, we do not observe a significant amount of large temperature variations. The reason is that our applications highly utilized the system, unlike the multi-threaded benchmarks with much more variant

Table 5.4: Workload Characteristics for the Architectural Simulator

Workload	**Benchmarks**
1) 16-CPU	mesa*3, bzip2_program*3, crafty*2, eon_rushmeier*3, vortex1*2, sixtrack*3
2) 16-MIX	mcf*2, mesa, art110, sixtrack*2, equake, bzip2_program, eon_rushmeier*2, swim, applu, twolf, crafty, apsi, lucas
3) 14-CPU	mesa*2, bzip2_program*3, crafty*2, eon_rushmeier*2, vortex1*2, sixtrack*3
4) 14-MIX	mcf*2, mesa, art110, sixtrack*2, equake, eon_rushmeier*2 swim, twolf, crafty, apsi, lucas

execution profile.

Next, we compare the performance of the temperature management techniques on the 16-core architecture. As in Chapter 2, we use the Fair Speedup Metric (FS) from [14] and [61]. Figure 5.21 provides the performance for each workload and policy, as well as the average case for the 16-core architecture. PTB increases the performance by over 3% in comparison to P-DVS and by over 5% in comparison to R-DVS. PTB achieves the same performance as P-Mig, while reducing the hot spots, as described earlier. Note that, on a single threaded system, the performance benefit of PTB over P-Mig diminishes, as PTB is a policy that is specifically designed for optimizing multi-threaded system performance.

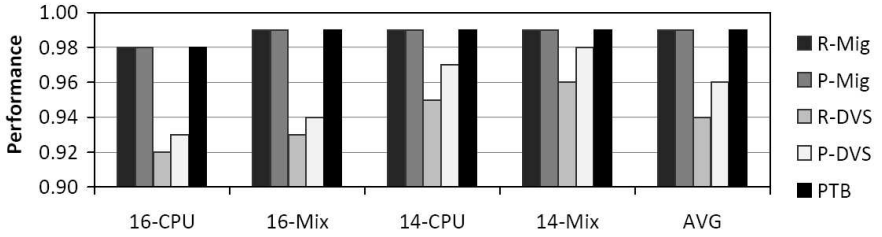

Figure 5.21: Performance of Policies on the 16-Core Architecture

We also run simulations where the ARMA predictor is used for predicting IPC (as described in Section 5.4) on the 16-core system. In this case, the proactive balancing policy utilizes the IPC predictions (referred to as PTB_IPC). In other words, prediction of high IPC is considered equivalent to a forecast of high power consumption. Therefore, the high-IPC threads are allocated to cooler locations. In the 16-core ($4x4$) MPSoC, the corner cores are typically cooler than other cores on the sides, and the cores in the center of the die are expected to be the hottest. How to order the cores in terms

of their susceptibility to hot spots during scheduling has been discussed in Chapter 2. Note that once the job with the highest IPC is allocated on one of the corner cores, the second highest-IPC job will be allocated on the opposite corner (across the diagonal) to minimize the possibility of hot spots—this way the policy avoids clustering the high power applications on neighboring cores. Thus, as PTB_IPC separates the high IPC jobs from each other and places the lowest power jobs in the central region of the die, it is effective in reducing the frequency of hot spots.

Figure 5.22 shows the thermal behavior achieved by PTB (temperature-based) and PTB_IPC. The two techniques result in very similar percentages of hot spots, whereas PTB_IPC has higher performance overhead due to more frequent migrations. PTB_IPC reacts to changes in IPC, which are not always reflected to the temperature profile due to the thermal time constants. The results show that, for a single-threaded system without temperature sensors, IPC is a reasonable metric to guide thermal management. Note that for other systems or workloads, PTB_IPC may result in higher percentage of hot spots as it does not consider the thermal interactions of neighboring units or the recent thermal history.

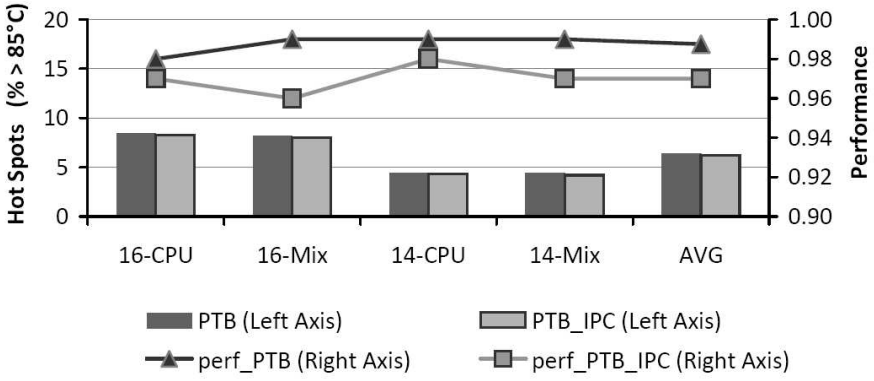

Figure 5.22: Thermal Results for ARMA IPC Predictor

We observe that in single-threaded MPSoCs, DVS has better results than job allocation policies (migration or balancing) in terms of reducing the hot spots. However, considering the performance cost of DVS is higher, it would be beneficial to design hybrid strategies combining DVS and job scheduling to achieve a more desirable temperature-performance trade-off. It should also be noted that DVS requires hardware support for the dynamic management of voltage, whereas the Proactive Temperature Balancing we propose can be performed by only modifying the OS dispatching policy.

Previously in this section we have seen that PTB accomplishes to reduce the frequency of harmful temperature events as much as DVS, while resulting in only a slight decrease in performance with respect to the default load balancing scheme. In multi-threaded MPSoCs, proactive management brings significantly more benefits in reducing hot spots and temperature variations in comparison to applying the equivalent policies in single-threaded systems. This is due to the fact that multi-threaded environment provides more opportunities for applying temperature-aware job allocation techniques without hurting performance.

5.7 Summary

This chapter has presented a proactive temperature management approach for multiprocessor system-on-chips (MPSoCs). The proposed management technique utilizes autoregressive moving average (ARMA) modeling to accurately predict future temperature on each core based solely on the previous measurements. We continuously monitor how well the ARMA model fits the current temperature using sequential probability ratio test (SPRT), and update the model if necessary. SPRT guarantees to achieve the fastest detection of changes in thermal dynamics. This way, the technique can adapt to changing workload conditions.

The *Proactive Temperature Balancing (PTB)* method proposed in the chapter utilizes the thermal forecast for dynamic allocation of threads, and reduces the thermal hot spots and temperature gradients significantly at very low performance impact. This technique does not require offline analysis or workload profiling, and achieves more accurate predictions under dynamically variant workload in comparison to methods that rely on offline analysis or longer training periods.

The chapter also has provided a detailed comparison of our ARMA/SPRT based approach to other prediction methods (e.g., exponential moving average, history predictor, and least squares predictor), as well as an experimental evaluation of both reactive and proactive thermal management approaches on single- and multi-threaded MPSoCs.

In the UltraSPARC T1 experiments, we have observed that our technique achieves 60% reduction in hot spot occurrences, 80% reduction in spatial gradients, and 75% reduction in thermal cycles on average in comparison to reactive thermal management. *Proactive Temperature Balancing* also results in better performance than the other migration and dynamic voltage/frequency scaling (DVS) policies. When integrated with

dynamic power management (DPM), *PTB* has similar chip-level energy savings with reactive thread migration and load balancing. Even though the DVS policies reduce the energy consumption of the cores significantly (i.e., by 21% in comparison to the 9% reduction of *PTB*), we expect results to be different for system-level energy. The reason is that DVS considerably increases the execution time. Consequently, the total energy consumption of the system may not benefit as much from DVS due to the prolonged execution times of memories and other components.

The text of Chapter 5 is in part a reprint of the material from the papers, *Ayse K. Coskun, Tajana Simunic Rosing, and Kenny Gross, "Utilizing Predictors for Efficient Thermal Management in Multiprocessor SoCs"*, in IEEE Transactions on CAD, 2009, and *Ayse K. Coskun, Tajana Simunic Rosing and Kenny Gross, "Proactive Temperature Balancing for Low Cost Thermal Management in MPSoCs"*, in Proceedings of International Conference on Computer-Aided Design (ICCAD), 2008. The dissertation author was the primary researcher and author, and the co-authors involved in the publications [20] and [15] directed, supervised, and assisted in the research which forms the basis for that material.

Chapter 6

Summary and Future Work Directions

As we enter the many-core era, where a single chip may have tens or hundreds of cores, capability of analyzing and managing temperature, energy, reliability, and performance at runtime will become even more important for finding the desirable operating points. Temperature-induced challenges are already a major concern for current MPSoCs, and in many-core systems resolving these challenges at design time will not be possible due to high cost and design complexity.

Conventional dynamic thermal management methods, such as thread migration or clock-gating, sacrifice performance to handle thermal emergencies. In addition to disrupting the performance when the demand to the system is the highest, such techniques do not reduce and balance the temperature as much as possible, but rather guarantee operating below a critical (unsafe) temperature value. However, reliability physics shows that avoiding thermal variations is as important as eliminating thermal hot spots, especially at the temperature ranges the current chips operate.

Most commercial multicore systems in the field today are not fully utilized for the majority of their lifetime. "Free cycles" in the MPSoCs provide opportunities for performance-efficient thermal management. On the other hand, having idle cycles can increase the temporal and spatial thermal variations. Therefore, we need to design management policies with a joint perspective on reliability, energy, and performance.

This thesis provides a framework for fast and accurate analysis of multicore system reliability, and develops performance-efficient thermal management strategies.

This chapter summarizes the main contributions of this work and points out the future research directions.

6.1 Fast Reliability Simulation Over Long Time Frames

An important step in designing efficient thermal management policies is to analyze the effects of design-time and runtime decisions on reliability. Architecture-level simulators are powerful tools to evaluate performance for new architectures or techniques. However, such simulators are computationally demanding and they take a long time to simulate full applications. For this reason, prior work has proposed techniques to select representative phases of benchmarks for fast performance analysis [57]. However, for meaningful temperature and reliability evaluation, we need to simulate full benchmarks, include the possible idle time slots and workload changes in the simulation, and run the simulation for real execution times equivalent to at least several minutes. Obviously, achieving this with the conventional architecture-level simulators is prohibitively time consuming, resulting in a simulation time of days or weeks at best.

This thesis proposes a novel simulation framework following the phase identification principle, but instead of selecting representative phases, we simulate complete benchmarks and identify the power consumption and performance characteristics of each phase. Then, this information for each target benchmark is stored in a database. When we are simulating multicore management or scheduling policies, the simulator queries the database with appropriate phase-IDs, collects the power and performance information, and uses the power information to compute the temperature and reliability. We show that our simulation framework is able to evaluate temperature and performance accurately within reasonable simulation times—i.e., we can simulate minutes of real-time execution in several hours, as opposed to architecture-level simulators that can take days or weeks.

We use this framework to analyze a number of power management, thermal management, and job scheduling policies in a multicore system. The results highlight the following key conclusions: (1) It is critical to consider thermal variations in addition to peak temperature effects to have a fair evaluation of the policies. (2) When some of the cores are idle, which is the case in most multicore systems, it is important to manage the idle cores without accelerating thermal cycling, which degrades reliability. (3) Reducing the unnecessary thread movements through intelligent workload allocation

is highly beneficial for improving both performance and reliability. (4) Understanding the thermal asymmetries (e.g., due to layout) in a multicore system is necessary to develop efficient management methods. Even in a homogeneous multicore system, all cores are not equal in terms of their thermal behavior. (5) Proactive techniques that make performance or temperature-aware management decisions without waiting for thermal emergencies help with both reliability and performance. We utilize these guidelines in designing efficient thermal management policies.

6.2 Performance-Efficient Thermal Management

A significant part of this thesis focuses on developing temperature-aware job scheduling methods. For a system with a known set of jobs, this thesis formulates and solves an integer linear program (ILP) that balances and minimizes the temperature on chip as much as possible. The static solution can be used for embedded systems with highly predictable workloads, and also for setting a baseline for dynamic management methods.

In addition to the ILP that minimizes both hot spots and gradients, we also implement other ILPs with various energy and thermal objectives for comparison. A significant conclusion of this comparison is that optimizing for energy does not guarantee effective management of hot spots and gradients. Contrarily, energy management benefits from clustering workloads to achieve longer idle time slots that can be utilized by switching to an ultra-low power saving mode, even though such an allocation is likely to increase temperature and cause hot spots. Therefore, management policies with temperature-specific objectives are needed to reduce the adverse affects of temperature.

In most systems workload varies at runtime, making it highly unlikely to successfully optimize the job schedule at design-time. This thesis discusses how to manage temperature dynamically with low-cost online techniques. An important feature of dynamic thermal management policies is to have adaptation capability to varying workload, system, or environmental conditions. We achieve this by using real-time feedback from the sensors available in the system for all the dynamic techniques we propose.

The thesis discusses two dynamic management solutions in particular: (1) *Adaptive-Random*: This policy changes each core's likeliness to receive workload based on the core's recent thermal history. It is able to balance the temperature, reducing both the hot spots and thermal variations at negligible performance cost. (2) *Online Learning*: Ther-

mal/power management policies have different strengths that makes them most efficient for specific workload conditions. *Online Learning* enables us to select the best fitting policy for the current workload by evaluating the thermal behavior and performance of a given set of "expert" policies. We show that *Adaptive-Random* and *Online Learning* can both significantly improve the thermal profile. Another important conclusion of our work is that temperature-aware scheduling is orthogonal to other power management or back-up thermal management strategies. In other words, workload scheduling approaches such as *Adaptive-Random* can be integrated with other management methods to improve the thermal behavior and performance simultaneously.

Previously proposed dynamic thermal management strategies are generally reactive in nature; that is, they take action after a thermal phenomenon occurs. A novel contribution of this thesis is proactive thermal management. Instead of waiting for a core to reach a critical temperature, we show that we can forecast temperature into the near future, and use this forecast to prevent hot spots and variations proactively. We use autoregressive moving average (ARMA) model for predicting temperature and show that the predictions are highly accurate. The main principles behind ARMA are learning the time-series temperature signal, modeling it with a polynomial equation (using an automated flow), and then predicting the future temperature using this model. If the workload dynamics change, we detect the change using a likelihood ratio test, and form a new predictor model to maintain the accuracy of the forecast.

Proactive Temperature Balancing (PTB) utilizes the forecast to balance the temperature among the cores, or move threads away from cores getting hotter. In this way we achieve a better thermal profile and also better performance, as we do not have to take emergency actions such as stalling or slowing down execution. Our proactive management method does not require any offline analysis when forecasting or managing the allocation of threads. The results achieved by *PTB* are remarkable. In the experiments performed on the UltraSPARC T1, we have observed that our technique achieves 60% reduction in hot spot occurrences, 80% reduction in spatial gradients and 75% reduction in thermal cycles in average, in comparison to reactive thermal management, while also improving performance. We also show how to predict workload for systems without thermal sensors, and use the workload prediction instead of temperature for proactive workload scheduling.

Learning the system and workload behavior at runtime and adapting the man-

agement policy will be even more important for more complex many-core systems. When the number of cores increase, such self-management capabilities will be critical to enable the systems to meet the performance demands, while at the same time reducing energy and keeping a balanced and low thermal profile. Similarly, proactive techniques that analyze current and future conditions, and act without waiting for unwanted events will ease the design of reliable systems.

6.3 Future Directions

6.3.1 Temperature Modeling and Management in 3D Systems

Technology scaling has caused the feature sizes to shrink continuously, whereas interconnects, unlike transistors, have not followed the same trend. In the nanometer era, a larger portion of the total chip capacitance comes from the interconnects. With the introduction of vias and repeaters to compensate for the performance loss of the long wires, the interconnect power consumption rises dramatically [36].

One solution to the rising power consumption in interconnects is 3D stacking [7], which reduces the length of the communication lines through vertical integration of circuit blocks. However, 3D stacking substantially increases power density and thermal resistances due to the placement of computational units on top of each other. High power densities are already a major concern in 2D circuits [3], and in 3D systems the problem is even more severe [7, 50]. The 3D stacked systems exacerbate temperature-induced problems, leading to degraded performance and reliability if not handled properly.

In 3D systems, chip cross-sectional power density increases with the number of vertically stacked circuit layers [75]. 3D integration complicates the implementation of dynamic thermal management techniques because of the heat transfer between vertically adjacent units and the heterogeneous cooling efficiencies of different layers (e.g., the components closer to the heat sink cool down easier than those further away). Therefore, traditional 2D thermal management policies are not sufficient to optimize the temperature profile of multicore 3D systems.

Our recent work [18] evaluates existing thermal management policies for 2D chips in terms of their thermal management capabilities in 3D systems. Based on the analysis, we propose an extension of *Adaptive-Random*, called *Adapt3D*, which takes into account the inherent imbalance in the cooling efficiencies of different layers as well as the location

of the cores for temperature-aware job scheduling. Such location impact is significant especially for 3D systems with more than two layers. *Adapt3D* provides a significant reduction on the frequency of hot spots, spatial gradients and thermal cycles. In fact, it achieves similar results to DVS in the optimization of thermal profiles, while the performance cost is kept to a minimum. When combined with DVS, *Adapt3D* improves the reduction of hot spots by an additional 20%-40% in comparison to performing only DVS, and reduces the performance cost.

In a recent design space exploration of the cost and temperature of 3D systems [16], we show that as technology scaling continues to $45nm$ and below, it may not be feasible to stack many layers in 3D systems with conventional cooling. This result is due to both the expected increase in power density in new technology nodes and the adverse thermal effects of stacking. 3D integration for many-core systems in $45nm$ and below will likely require more efficient cooling infrastructures, such as liquid cooling. An open research area in this field is to develop thermal modeling and management methods for systems with such novel cooling infrastructures.

6.3.2 Managing Parallel Workloads in Multicore Architectures

With the design trend moving to many-core systems, the programming paradigm also shifts to parallel applications to be able to exploit the benefits of multicore design more effectively. This thesis has investigated both single- and multi-threaded applications, but parallel applications are beyond the focus of our work.

For parallel applications with limited inter-thread interactions, the proposed methods would still be applicable. For example, recent research on multicore caching has focused on reducing those interactions [33], and in the extreme, such techniques could be configured to make the shared caches act as private caches. However, for any parallel applications requiring a considerable amount of time devoted to synchronization, the management policies would not be able to capture and effectively manage the parallel program behavior.

Basic guidelines derived in this work will continue to be useful for designing management approaches for parallel applications, such as: (1) Paying attention to the thermal asymmetries of the MPSoC, (2) Reducing the number of unnecessary thread movements, and (3) Applying adaptive and proactive techniques that can learn the system/workload conditions and make intellectual decisions. One of the open areas in par-

allel programming is analyzing which performance or design-related parameters (i.e., in addition to physical variables such as temperature) would be relevant for understanding the application behavior and for guiding the dynamic management policies.

The text of Section 6.3.1 is in part a reprint of the material from the paper, *Ayse K. Coskun, Tajana Simunic Rosing, Jose Ayala, David Atienza and Yusuf Leblebici, "Dynamic Thermal Management in 3D Multicore Architectures"*, in Proceedings of Design Automation and Test in Europe (DATE), 2009. The dissertation author was the primary researcher and author, and the co-authors involved in the publication [18] directed, supervised, and assisted in the research which forms the basis for that material.

The text of Section 6.3.1 is in part a reprint of the material from the paper, *Ayse K. Coskun, Andrew B. Kahng and Tajana Simunic Rosing, "Temperature- and Cost-Aware Design of 3D Multiprocessor Architectures"*, in Proceedings of Euromicro Conference on Digital System Design (DSD), 2009. The dissertation author was the primary researcher and author, and the co-authors involved in the publication [16] directed, supervised, and assisted in the research which forms the basis for that material.

Bibliography

[1] A. H. Ajami, K. Banerjee, and M. Pedram. Modeling and Analysis of Nonuniform Substrate Temperature Effects on Global ULSI Interconnects. *IEEE Transactions on CAD*, 24(6):849–861, June 2005.

[2] A. R. Alameldeen and D. A. Wood. IPC Considered Harmful for Multiprocessor Workloads. *IEEE Micro*, 26(4):8–17, 2006.

[3] D. Atienza, P. D. Valle, G. Paci, F. Poletti, L. Benini, G. D. Micheli, and J. M. Mendias. A Fast HW/SW FPGA-Based Thermal Emulation Framework for Multi-Processor System-on-Chip. In *Design Automation Conference (DAC)*, pages 618–623, 2006.

[4] L. Benini, A. Bogliolo, and G. D. Micheli. A Survey of Design Techniques for System-Level Dynamic Power Management. *IEEE Trans. Very Large Scale Integr. Syst.*, 8(3):299–316, 2000.

[5] M. V. Biesbrouck, T. Sherwood, and B. Calder. A Co-Phase Matrix to Guide Simultaneous Multithreading Simulation. In *IEEE International Symposium on Performance Analysis of Systems and Software*, pages 45–56, 2004.

[6] N. L. Binkert, E. G. Hallnor, and S. K. Reinhardt. Network-Oriented Full-System Simulation Using M5. In *Workshop on Computer Architecture Evaluation using Commercial Workloads (CAECW)*, 2003.

[7] B. Black, M. Annavaram, N. Brekelbaum, J. DeVale, L. Jiang, G. H. Loh, D. McCaule, P. Morrow, D. W. Nelson, D. Pantuso, P. Reed, J. Rupley, S. Shankar, J. Shen, and C. Webb. Die Stacking (3D) Microarchitecture. In *MICRO 39: Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 469–479, 2006.

[8] P. Bose. Power-Efficient Microarchitectural Choices at the Early Design Stage. In *Keynote Address, Workshop on Power-Aware Computer Systems*, 2003.

[9] S. Bouchenak and D. Hagimont. Pickling Threads State in the Java System. In *Technology of Object-Oriented Languages and Systems (TOOLS) Europe*, 2000.

[10] S. Boyd and L. Vandenberghe. Convex Optimization. *Cambridge University Press*, 2004.

[11] D. Brooks and M. Martonosi. Dynamic Thermal Management for High-Performance Microprocessors. In *Proceedings of International Symposium on High Performance Computer Architecture (HPCA)*, pages 171–182, 2001.

[12] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: A Framework for Architectural-Level Power Analysis and Optimizations. In *International Symposium on Computer Architecture (ISCA)*, pages 83–94, 2000.

[13] D. C. Burger and T. M. Austin. The SimpleScalar Tool Set, Version 2.0. Technical Report CS-TR-97-1342, University of Wisconsin, Madison, June 1997.

[14] J. Chang and G. S. Sohi. Cooperative Cache Partitioning for Chip Multiprocessors. In *International Conference on Supercomputing (ICS)*, pages 242–252, 2007.

[15] A. K. Coskun, T. Rosing, and K. Gross. Proactive Temperature Balancing for Low-Cost Thermal Management in MPSoCs. In *International Conference on Computer-Aided Design (ICCAD)*, pages 250–257, 2008.

[16] A. K. Coskun, T. Rosing, and A. B. Kahng. Temperature- and Cost-Aware Design of 3D Multiprocessor Architectures. In *Euromicro Conference on Digital System Design (DSD)*, pages 183–190, 2009.

[17] A. K. Coskun, T. Rosing, and K. Whisnant. Temperature Aware Task Scheduling in MPSoCs. In *Design Automation and Test in Europe (DATE)*, pages 1659–1664, 2007.

[18] A. K. Coskun, T. S. Rosing, J. Ayala, D. Atienza, and Y. Leblebici. Dynamic Thermal Management in 3D Multicore Architectures. In *DATE*, pages 1410–1415, 2009.

[19] A. K. Coskun, T. S. Rosing, and K. Gross. Temperature Management in Multiprocessor SoCs Using Online Learning. In *Design Automation Conference (DAC)*, pages 890–893, 2008.

[20] A. K. Coskun, T. S. Rosing, and K. Gross. Utilizing Predictors for Efficient Thermal Management in Multiprocessor SoCs. *IEEE Transactions on CAD (accepted for publication)*, 2009.

[21] A. K. Coskun, T. S. Rosing, K. A. Whisnant, and K. C. Gross. Static and Dynamic Temperature-Aware Scheduling for Multiprocessor SoCs. *IEEE Transactions on VLSI*, 16(9):1127–1140, Sept. 2008.

[22] A. K. Coskun, R. Strong, D. Tullsen, and T. S. Rosing. Evaluating the Impact of Job Scheduling and Power Management on Processor Lifetime for Chip Multiprocessors. In *SIGMETRICS/Performance–Joint International Conference on Measurement and Modeling of Computer Systems*, pages 169–180, 2009.

[23] G. Dhiman and T. Rosing. Dynamic Voltage Frequency Scaling for Multi-Tasking Systems Using Online Learning. In *International Symposium on Low Power Electronics and Design (ISLPED)*, pages 207–212, 2007.

[24] J. Donald and M. Martonosi. Techniques for Multicore Thermal Management: Classification and New Exploration. In *ISCA*, pages 78–88, 2006.

[25] Y. Freund, R. E. Schapire, Y. Singer, and M. K. Warmuth. Using and Combining Predictors That Specialize. In *STOC '97: Annual ACM Symposium on Theory of Computing*, pages 334–343, 1997.

[26] M. Gomaa, M. D. Powell, and T. N. Vijaykumar. Heat-and-Run: Leveraging SMT and CMP to Manage Power Density through the Operating System. In *Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 260–270, 2004.

[27] K. Gross, K. Whisnant, and A. Urmanov. Electronic Prognostics through Continuous System Telemetry. In *60th Meeting of the Society for Machinery Failure Prevention Technology (MFPT)*, pages 53–62, April 2006.

[28] K. C. Gross and K. E. Humenik. Sequential Probability Ratio Test for Nuclear Plant Component Surveillance. *Nuclear Technology*, 93(2):131–137, Feb 1991.

[29] S. Heo, K. Barr, and K. Asanovic. Reducing Power Density through Activity Migration. In *ISLPED*, pages 217–222, 2003.

[30] M. Herbster and M. K. Warmuth. Tracking the Best Expert. In *International Conference on Machine Learning*, pages 286–294, 1995.

[31] Intel Pentium 4 Processor in the 423-Pin Package Thermal Design Guidelines. Technical Report 249203-001, Intel, November 2000.

[32] C. Isci, G. Contreras, and M. Martonosi. Live, Runtime Phase Monitoring and Prediction on Real Systems with Application to Dynamic Power Management. In *MICRO 39*, pages 359–370, 2006.

[33] R. Iyer, L. Zhao, F. Go, R. Illikkal, S. Makineni, D. Newell, Y. Solihin, L. Hsu, and S. Reinhardt. QoS Policies and Architecture for Cache/Memory in CMP Platforms. In *ACM SIGMETRICS*, pages 25–36, 2007.

[34] Failure Mechanisms and Models for Semiconductor Devices, JEDEC Publication JEP122C. http://www.jedec.org.

[35] J. A. Kahle, M. N. Day, H. P. Hofstee, C. R. Johns, T. R. Maeurer, and D. Shippy. Introduction to the Cell Multiprocessor. *IBM Journal of Research & Development*, 49(4/5):589–604, July/September 2005.

[36] P. Kapur, G. Chandra, and K. Saraswat. Power Estimation in Global Interconnects and Its Reduction Using a Novel Repeater Optimization Methodology. In *DAC*, pages 461–466, 2002.

[37] A. Karlin, M. Manesse, L. McGeoch, and S. Owicki. Competitive Randomized Algorithms for Nonuniform Problems. *Algorithmica*, pages 542–571, 1994.

[38] P. Kongetira, K. Aingaran, and K. Olukotun. Niagara: A 32-way multithreaded SPARC processor. *IEEE Micro*, 25(2):21–29, 2005.

[39] H. Kufluoglu and M. A. Alam. A Computational Model of NBTI and Hot Carrier Injection Time-Exponents for MOSFET Reliability. *Journal of Computational Electronics*, 3 (3):165–169, Oct. 2004.

[40] A. Kumar, L. Shang, L.-S. Peh, and N. K. Jha. HybDTM: A Coordinated Hardware-Software Approach for Dynamic Thermal Management. In *DAC*, pages 548–553, 2006.

[41] R. Kumar, D. M. Tullsen, and N. P. Jouppi. Core Architecture Optimization for Heterogeneous Chip Multiprocessors. In *PACT*, pages 23–32, 2006.

[42] R. Kumar, V. Zyuban, and D. M. Tullsen. Interconnections in Multi-Core Architectures: Understanding Mechanisms, Overheads and Scaling. In *ISCA*, pages 408–419, 2005.

[43] E. Kursun, C.-Y. Cher, A. Buyuktosunoglu, and P.Bose. Investigating the Effects of Task Scheduling on Thermal Behavior. In *3rd Workshop on Temperature-Aware Computer Systems (TACS)*, 2006.

[44] C. J. Lasance. Thermally Driven Reliability Issues in Microelectronic Systems: Status-quo and Challenges. *Microelectronics Reliability*, 43:1969–1974, 2003.

[45] A. Leon, L. Jinuk, K. Tam, W. Bryg, F. Schumacher, P. Kongetira, D. Weisner, and A. Strong. A Power-Efficient High-Throughput 32-Thread SPARC Processor. *Journal of Solid State Circuits*, pages 7–16, 2006.

[46] L. Ljung, editor. *System Identification (2nd ed.): Theory for the User*. Prentice Hall PTR, 1999.

[47] Lp_solve. http://lpsolve.sourceforge.net/5.5/.

[48] R. McDougall, J. Mauro, and B. Gregg. Solaris Performance and Tools. *Sun Microsystems Press*, 2006.

[49] H. Nguyen. Multilevel Interconnect Reliability on the Effects of Electro-Thermomechanical Stresses. Ph.D. Dissertation, University of Twente, Netherlands, 2004.

[50] K. Puttaswamy and G. H. Loh. Thermal Herding: Microarchitecture Techniques for Controlling Hotspots in High-Performance 3D-Integrated Processors. In *HPCA*, pages 193–204, 2007.

[51] P. Rong and M. Pedram. Power-Aware Scheduling and Dynamic Voltage Setting for Tasks Running on a Hard Real-Time System. In *Asia and South Pacific Design Automation Conference (ASPDAC)*, pages 473–478, 2006.

[52] T. S. Rosing, K. Mihic, and G. D. Micheli. Power and Reliability Management of SoCs. *IEEE Transactions on VLSI*, 15(4):391–403, April 2007.

[53] M. Ruggiero, A. Guerri, D. Bertozzi, F. Poletti, and M. Milano. Communication-Aware Allocation and Scheduling Framework for Stream-Oriented Multi-Processor System-on-Chip. In *DATE*, pages 3–8, 2006.

[54] K. Sankaranarayanan, S. Velusamy, M. Stan, and K. Skadron. A Case for Thermal-Aware Floorplanning at the Microarchitectural Level. *The Journal of Instruction-Level Parallelism*, 7, 2005.

[55] M. Santarini. Thermal Integrity: A Must for Low-Power IC Digital Design. *Electronics, Design, Strategy, News (EDN)*, pages 37–42, Sept. 2005.

[56] L. Seiler, D. Carmean, E. Sprangle, T. Forsyth, M. Abrash, P. Dubey, S. Junkins, A. Lake, J. Sugerman, R. Cavin, R. Espasa, E. Grochowski, T. Juan, and P. Hanrahan. Larrabee: A Many-Core x86 Architecture for Visual Computing. In *ACM SIGGRAPH*, pages 1–15, 2008.

[57] T. Sherwood, G. H. E. Perelman, and B. Calder. Automatically Characterizing Large Scale Program Behavior. In *ASPLOS*, pages 45–57, 2002.

[58] Y.-H. Shih and J.-G. Hwu. An On-Chip Temperature Sensor by Utilizing a MOS Tunneling Diode. *IEEE Electron Device Letters*, 22(6), 2001.

[59] K. Skadron, M. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan. Temperature-Aware Microarchitecture. In *ISCA*, pages 2–13, 2003.

[60] SLAMD Distributed Load Engine. www.slamd.com.

[61] J. Smith. Characterizing Computer Performance with a Single Number. *Commun. ACM*, 31(10):1202–1206, 1988.

[62] J. Srinivasan, S. V. Adve, P. Bose, and J. A. Rivers. The Case for Lifetime Reliability-Aware Microprocessors. In *ISCA*, pages 276–287, 2004.

[63] J. Srinivasan, S. V. Adve, P. Bose, and J. A. Rivers. The Impact of Technology Scaling on Lifetime Reliability. In *IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 177–186, 2004.

[64] J. Srinivasan, S. V. Adve, P. Bose, and J. A. Rivers. Lifetime Reliability: Toward an Architectural Solution. *IEEE Micro*, 25(3):70–80, 2005.

[65] K. Stavrou and P. Trancoso. Thermal-Aware Scheduling for Future Chip Multiprocessors. *EURASIP Journal on Embedded Systems*, 2007.

[66] H. Su, F. Liu, A. Devgan, E. Acar, and S. Nassif. Full-Chip Leakage Estimation Considering Power Supply and Temperature Variations. In *ISLPED*, pages 78–83, 2003.

[67] D. Tarjan, S. Thoziyoor, and N. P. Jouppi. CACTI 4.0. Technical Report HPL-2006-86, HP Laboratories Palo Alto, 2006.

[68] M. Tremblay and S. Chaudhry. A Third-Generation 65nm 16-Core 32-Thread Plus 32-Scout-Thread CMT SPARC Processor. In *ISSCC*, pages 82–83, 2008.

[69] V. V. Vazirani. *Approximation Algorithms*. Springer: Berlin, Germany, 2003.

[70] R. Viswanath, V. Wakharkar, A. Watwe, and V. Lebonheur. Thermal Performance Challenges from Silicon to Systems. *Intel Technology Journal*, Q3, 2000.

[71] A. Wald and J. Wolfowitz. Optimum Character of the Sequential Probability Ratio Test. *Ann. Math. Stat.*, 19:326, 1948.

[72] Q. Wu, M. Martonosi, D. W. Clark, V. J. Reddi, D. Connors, Y. Wu, J. Lee, and D. Brooks. A Dynamic Compilation Framework for Controlling Microprocessor Energy and Performance. In *MICRO 38*, pages 271–282, 2005.

[73] I. Yeo, C. C. Liu, and E. J. Kim. Predictive Dynamic Thermal Management for Multicore Systems. In *DAC*, pages 734–739, June 2008.

[74] Y. Yu and V. Prasanna. Energy-Balanced Task Allocation for Collaborative Processing in Wireless Sensor Networks. *Mobile Networks and Applications*, 10:115–131, 2005.

[75] C. Zhu, Z. Gu, L. Shang, R. P. Dick, and R. Joseph. Three-Dimensional Chip-Multiprocessor Run-Time Thermal Management. *IEEE Transactions on CAD*, 27(8):1479–1492, August 2008.